



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

1981

A cross compiler and programming support system for the HP41CV.

Richmann, James Norman

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/20480>

Downloaded from NPS Archive: Calhoun



<http://www.nps.edu/library>

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIF, 93940

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A CROSS COMPILER AND PROGRAMMING SUPPORT
SYSTEM FOR THE HP41CV CALCULATOR

By

James Norman Richmann

September 1981

Thesis Advisors:

S. H. Parry
R. H. Shudde

Approved for public release; distribution unlimited.

T202784

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Cross Compiler and Programming Support System for the HP41CV Calculator		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1981
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James Norman Richmann		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE September 1981
		13. NUMBER OF PAGES 242
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Calculator, Cross Compiler, HP41CV Programmable Calculator, Optical Bar Code		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) With growing Army-wide use of programmable calculators, a system is needed to support the programming and testing of calculator software. This thesis provides a FORTRAN IV program to enable an operations research analyst to more efficiently write and document HP41CV calculator programs. Optical bar code readable by the HP41CV is generated by the program. Also given is an		

IBM EXEC II program which provides an interactive programming environment including on-line, self contained instructions. To illustrate the use of the system and the quality of the finished bar code and calculator program listings, examples are given including single variable statistics and linear programming. A final example provides a set of short utility routines which illustrate how programs can be developed for use in a calculator read-only-memory.

Approved for public release, distribution unlimited.

A Cross Compiler and Programming Support
System for the HP41CV Calculator

by

James Norman Richmann
Captain, United States Army
B.S., Iowa State University, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the
NAVAL POSTGRADUATE SCHOOL
September, 1981

ABSTRACT

With growing Army-wide use of programmable calculators, a system is needed to support the programming and testing of calculator software. This thesis provides a Fortran IV program to enable an operations research analyst to more efficiently write and document HP41CV calculator programs. Optical bar code readable by the HP41CV is generated by the program. Also given is an IBM EXEC II program which provides an interactive programming environment including on-line, self contained instructions. To illustrate the use of the system and the quality of the finished bar code and calculator program listings, examples are given including single variable statistics and linear programming. A final example provides a set of short utility routines which illustrate how programs can be developed for use in a calculator read-only-memory.

TABLE OF CONTENTS

I.	INTRODUCTION	- - - - -	8
II.	THE PROGRAMMING ENVIRONMENT	- - - - -	23
A.	CHAPTER OVERVIEW	- - - - -	23
B.	STRUCTURED PROGRAMMING WITH THE HP41CV	- - - - -	24
1.	The Need for Structure	- - - - -	24
2.	Fundamental Limitations of Calculators	- - -	25
3.	Modular Design	- - - - -	27
4.	Control of Program Flow	- - - - -	28
5.	Clarification of Program Structure	- - - - -	29
6.	Data Types and Indirect Addressing	- - - - -	31
C.	ADDITIONAL CRITERIA FOR PROGRAM EVALUATION	- - -	33
1.	User Friendliness	- - - - -	35
2.	Execution Speed	- - - - -	37
D.	A PROGRAMMING SUPPORT SYSTEM	- - - - -	39
1.	A Cross Compiler and Bar Code Generator	- - -	40
2.	A Calculator Emulator	- - - - -	41
3.	A Higher Level Language Compiler	- - - - -	42
APPENDIX A:	SINGLE VARIABLE STATISTICS EXAMPLE	- - - - -	44
APPENDIX B:	LINEAR PROGRAMMING EXAMPLE	- - - - -	72
APPENDIX C:	SUBROUTINES FOR READ ONLY MEMORY	- - - - -	107
APPENDIX D:	THE CROSS COMPILER PROGRAM AND COMMAND		
	PROCESSOR	- - - - -	128

LIST OF REFERENCES - - - - - 237

INITIAL DISTRIBUTION LIST - - - - - 238

LIST OF FIGURES

1.	Programming Environment Command Menu - - - - -	12
2.	List of Commands - - - - -	13
3.	On-Line Introductory Material - - - - -	15
4.	Components of Program Documentation - - - - -	21
5.	Example Program to Add n Numbers - - - - -	30
6.	Program to Recall an Element of a Matrix - - - - -	34

I. INTRODUCTION

For the Army to fight effectively in a resource scarce environment, the quantitative decision making techniques of operations research are important skills for Army staff officers. Staff officers are expected to be able to put numbers in their estimates when briefing commanders. They are expected to be able to measure and evaluate complex operations and subordinate units. They are expected to be frugal managers of time and money. And above all, staff officers must be able to apply sound, quantified reasoning in planning how to win the air-land battle.

The use of hand-held programmable calculators by Army staff officers has the potential for improving the use of quantitative decision making techniques throughout the Army. Faster and more accurate than paper and pencil, the calculator is less expensive and more portable than larger computers. Even when compared to the latest micro-computer systems or to portable terminals used for distributed data processing, the hand-held programmable calculator offers advantages in cost, reliability, power consumption and emission of electromagnetic radiation. Hand-held programmable

calculators have already been successfully used by soldiers in the field for applications in artillery fire direction, surveying, and navigation. In addition, large numbers of Army officers own their own pocket calculators and routinely use them for staff planning and reporting functions.

In January of 1981 the U. S. Army Command and General Staff College at Fort Leavenworth, Kansas selected a programmable calculator for the Combined Arms and Services Staff School (CAS³.) Using both resident and non-resident instruction, this course is designed to teach all Army captains staff techniques and procedures. As a significant part of the curriculum, the students are introduced to subjects such as statistics and regression, decision theory, combat modeling and linear programming. Considering the large number of officers projected to attend this course in future years, this course represents the most widespread training in operations research techniques ever attempted by the Army. The decision to provide a sophisticated calculator to these students on an experimental basis was made for two fundamental reasons. First, the availability of a calculator with immediate field utility should motivate the student to apply the quantitative techniques as compared to the student who would be forced to do all calculations by

hand. Second, the power of the calculator permits classroom discussion of techniques such as linear programming and regression which are very difficult and time consuming to perform manually.

This thesis documents the author's work to support the use of a calculator in the Combined Arms and Services Staff School. Initially, the intent was to produce a series of lesson materials incorporating the use of the calculator on a series of operations research topics which have immediate application for the Army division level staff officer. Instead, the work accomplished focused on the design and construction of a system to make the programming and testing of calculator programs easier and more efficient. Except for the introduction, this thesis is written for the person wishing to implement the programming support system described. The implementor must have a detailed knowledge of the instruction set and programming characteristics of the HP41CV calculator as described in Wickes [Ref. 1: pp. 6-20]. For the eventual user of the system, as compared to the implementor, the system itself provides on-line documentation on how to use the system and what commands and options are available. Figure 1 shows the command menu displayed on the terminal screen by this interactive program;

Figure 2 gives a more detailed explanation of each of the commands; and Figure 3 displays the on line introductory material that is provided to new users of the system. For the user, a knowledge of the information contained in the calculator owner's handbook [Ref. 2] is sufficient to begin writing calculator programs using the support system described.

The calculator selected by the Command and General Staff College, the Hewlett-Packard HP41CV, typifies the state of the art in off-the-shelf calculator technology. While not without disadvantages, this calculator was selected because of its power and features which make it easier for Army staff officers to use. First and most important of these features is the ability of the calculator to manipulate alphabetic characters in addition to numeric data. The calculator can display the name of a variable when input data is required or label output when the calculation is completed. With this feature, the calculator helps the user know what data to input or what action to take next. It also helps alleviate the need for constant reference to printed instructions which are difficult to use under field conditions.

HP41C CROSS COMPILER Program name EDITION=17 SEP 81
 SELECT DESIRED COMMAND FROM THE FOLLOWING:

PF-KEY	COMMAND	CODE	ACTION TAKEN BY PROGRAMMING	COMMAND SYSTEM
PF13	STOP	S	GETS YOU OUT OF THE HP41C CROSS COMPILER	
PF14	HELP	H	SHORT EXPLANATION OF HOW TO USE THE CROSS COMPILER	
PF15	ENTER	E	INTERACTIVE PROGRAM ENTRY (NO FILE CREATED)	
PF16	BAR	B	SUBMIT JOB FOR PHYSICAL PRODUCTION OF BAR CODE	
PF17	NEW	N	BEGIN WORK ON A NEW PROGRAM OR NAMED SUBROUTINE	
PF18	DIRECT	D	DIRECTORY OF COMMANDS	
PF19	LIST	L	DISPLAY NAMES OF HP41C PROGRAMS ON DISK	
PF20	OCOMP	O	OFFLINE COMPILE AND AUTO GENERATION OF BAR CODE	
PF21	PRINT	P	PRODUCE A HARDCOPY PRINTED LISTING OF THE PROGRAM	
PF22	* COMP	*	RESERVED A SOURCE LISTING USE BY HP41 EMMULATOR	
PF23	XEDIT	X	EDIT THE PROGRAM USING THE CMS FULL-SCREEN EDITOR	
PF24	ERASE		ERASE THE SOURCE FILE, LISTING FILE AND TEXT FILE	
	CMS		ALLOWS EXECUTION OF ANY VALID CMS COMMAND	
	CP		ALLOWS EXECUTION OF ANY VALID CP COMMAND	

INPUT COMMAND:

Figure 1: Programming Environment Command Menu

PF-KEY	CMD	CODE	ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM
PF13	STOP	S	THIS COMMAND IS USED WHEN YOU WISH TO STOP PROCESSING HP41C PROGRAMS AND RETURN TO CMS. IF YOU ARE EXECUTING A FUNCTION THAT WAS INVOKED FROM THE COMMAND MENU, IN MOST CASES PF13 WILL RETURN YOU TO THE MENU, AND BY PRESSING PF13 AGAIN YOU WILL RETURN TO CMS.
PF14	HELP	H	THIS COMMAND IS USED TO DISPLAY THE DETAILED EXPLANATION OF THE MENU COMMAND PROCESSOR AND ITS AVAILABLE COMMANDS. IF YOU HAVE QUESTIONS ABOUT THE PROCESS OF WRITING ACTUAL HP41C PROGRAMS YOU SHOULD CONSULT THE HP41 OWNER'S HANDBOOK.
PF15	ENTER	E	THIS COMMAND IS USED TO ENTER A PROGRAM USING THE CROSS-COMPILER IN AN INTERACTIVE MODE. THE ADVANTAGE OF THIS MODE IS THAT ANY SYNTACTICAL ERRORS IN THE HP41C PROGRAM ARE IMMEDIATELY IDENTIFIED BY THE CROSS-COMPILER AND AN ERROR MESSAGE IS SHOWN ON THE SCREEN. THE DISADVANTAGE IS THAT THE USER IS TOTALLY RESPONSIBLE FOR UPPER AND LOWER CASE BEING ENTERED PROPERLY.
PF16	BAR	B	THIS COMMAND IS USED ONCE THE HP41C PROGRAM IS WRITTEN AND COMPILED WITHOUT ERRORS. IT SUBMITS A JOB TO MVS BATCH FOR THE PHYSICAL PRODUCTION OF THE BAR CODE.
PF17	NEW	N	THIS COMMAND IS USED TO DIRECT THE ATTENTION OF THE COMMAND PROCESSOR TO A NEW HP41 PROGRAM SOURCE FILE. WHEN USED TO INITIATE A NEW HP41C PROGRAMS, IT AUTOMATICALLY INSURES THAT A NEW FILE IS CREATED WITH FILETYPE "HP41" AND PROMPTS THE USER FOR THE PROGRAM TITLE WHICH IS THE MANDATORY FIRST LINE OF EVERY HP41C SOURCE CODE FILE.
PF18	DIREC	D	THIS COMMAND DISPLAYS THE FULL COMMAND MENU. IT HAS PRIMARY USE WHEN YOU FINISH AN OPERATION THAT FILLS THE SCREEN WITH TEXTUAL MATTER AND YOU RECEIVE ONLY THE PROMPT "INPUT COMMAND".

Figure 2: List of Commands

PF-KEY	CMD	CODE	ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM
PF19	LIST	L	THIS COMMAND DISPLAYS "FLIST" FOR THOSE HP41C PROGRAMS THAT ARE ACTIVE ON YOUR A DISK. FROM THIS LIST YOU CAN ERASE OLD PROGRAMS TO RELEASE DISK STORAGE, CHANGE THE NAME OF PROGRAMS, OR EXAMINE THE CONTENTS OF ANY PROGRAM.
PF20	OCOMP	C O	THIS COMMAND IS USED TO PRODUCE AN "OFFLINE" COMPILE. THE PROGRAM LISTING IS AUTOMATICALLY PRINTED IN HARD COPY ON THE HIGH SPEED PRINTER. IF THE COMPILE WAS WITHOUT ERROR THE BAR CODE IS AUTOMATICALLY PRODUCED.
PF21	PRINT	P	THIS COMMAND PRINTS A COPY OF THE "LISTING" FILE ON THE HIGH SPEED PRINTER. IF YOU WISH TO HAVE A PRINTED COPY OF THE SOURCE CODE WITHOUT THE CROSS-COMPILER'S FEEDBACK, IT IS BEST TO SIMPLY PRINT THE SOURCE CODE CMS FILE BY ISSUING THE CMS PRINT COMMAND.
PF22	GO	G	THIS COMMAND IS USED TO INVOKE THE HP41C EMMULATOR PROGRAM WHICH ALLOWS YOU TO TEST EXECUTION OF THE PROGRAM ON THE LARGE COMPUTER. THE EMULATION PROGRAM WILL EXECUTE THE PROGRAM EXACTLY AS YOUR CALCULATOR WOULD. THIS COMMAND HAS NOT BEEN IMPLEMENTED AS OF 17 SEP 81.
PF23	COMP	C	THIS COMMAND IS USED TO INVOKE THE CROSS COMPILER TO TRANSLATE AN HP41C PROGRAM WRITTEN ON CMS DISK IN SOURCE CODE FORM. AFTER THE COMPILER THE USER IS AUTOMATICALLY PLACED IN THE CMS BROWSE MODE FOR THE OUTPUT "LISTING" FILE THAT RESULTED FROM THE COMPILER.
PF24	XEDIT	X	THIS COMMAND IS USED TO INVOKE THE FULL-SCREEN EDITOR TO MAKE MODIFICATIONS TO THE HP41C SOURCE CODE FILE.

Figure 2 (Continued)

HP41C CROSS COMPILER COMMAND PROCESSOR

YOU ARE CURRENTLY EXECUTING A CMS EXEC FILE THAT MAKES IT EASY TO INVOKE THE HP41C CROSS COMPILER AND WRITE PROGRAMS USING CMS AND THE IBM 3278 DISPLAY TERMINAL. COMMON PROGRAMMING REQUIREMENTS SUCH AS EDITING CAN BE ACCOMPLISHED IN THREE WAYS:

- USING THE PROGRAMMED FUNCTION KEYS (PF KEYS)
- USING A SHORT COMMAND WORD
- USING A ONE OR TWO LETTER MNEMONIC CODE

THE COMMAND ACTIONS AND THEIR ASSOCIATED PF KEYS AND CODES ARE ALL GIVEN IN A DIRECTORY WHICH IS DISPLAYED WHEN THE COMMAND PROCESSOR IS WAITING FOR YOUR INPUT.

IN ORDER TO GO FROM A PROGRAM IN YOUR HEAD TO THE FINISHED BAR CODE THERE ARE THREE MAIN STEPS:

- (1) EDIT. THE PROGRAM MUST BE PREPARED AS INPUT TO THE CROSS COMPILER. THE EASIEST WAY TO DO THIS IS WITH THE CMS XEDIT FACILITY.
- (2) COMPILE. THE PROGRAM MUST BE PROCESSED BY THE CROSS-COMPILER. THE CROSS-COMPILER IS ACTUALLY A FORTRAN PROGRAM WHICH PRODUCES TWO CMS FILES AS OUTPUT. BOTH THESE FILES HAVE THE SAME NAME AS YOUR PROGRAM NAME, BUT HAVE DIFFERENT FILE TYPES. THE "LISTING" FILE SHOWS THE RESULTS OF THE COMPILE STEP INCLUDING ANY ERRORS AND THE "DATA" FILE IS A FILE OF ZERO'S AND ONE'S USED BY THE BAR CODE GENERATOR.
- (3) BAR. THE "DATA" FILE FROM THE COMPILE STEP IS USED AS INPUT TO PRODUCE THE ACTUAL BAR CODE. YOU SHOULD NEVER PERFORM THIS STEP UNTIL YOUR PROGRAM HAS SUCCESSFULLY COMPILED WITHOUT ERRORS. THIS STEP IS DONE BY THE BATCH PROCESSOR AND IT MAY TAKE SEVERAL HOURS TO GET YOUR FINISHED BAR CODE.

Figure 3: On-Line Introductory Material

A second important feature is the multiplicity of means by which programs can be entered into the calculator. Magnetic cards, read only memory, and optical bar code are all available and each has advantages depending on the situation. For the long term, read only memory offers the ability to retain very large programs (in excess of 8000 bytes) and the simplest and most reliable means of entering programs into the calculator under field conditions. For the short term, optical bar code offers the least expensive method of reproducing and distributing calculator software that has not been subject to extensive field testing. In addition, as shown in this thesis, the optical bar code can provide an important link between a main-frame computer and the hand-held calculator.

A third important feature of the HP41CV is its relatively large memory capacity as compared to programmable calculators such as the Texas Instruments TI-59. A large amount of memory permits the solution of larger, often more realistic problems than could previously be solved on a hand-held device. A demonstration program given in this thesis for linear programming is an example of an application where the full memory capability of the HP41CV is required to be able to solve realistic problems.

To take advantage of the calculator's unequalled economy and portability, the operations research analyst is challenged to overcome its limits of speed and memory capacity. The preparation of calculator software is as difficult, if not more so, than the preparation of software for larger computers. To accomplish the most possible with the handheld device, the calculator programmer is often forced to write programs which are very difficult to comprehend when examined by other programmers. As Dahl, Dijkstra and Hoare [Ref. 3: pp. 1-10] point out there are limits to human competence which interfere with the programming process. In the past, with less mature calculators which constrained the typical program to a few hundred program steps, these limits to human competence were neither as apparent nor as economically important as they are with the HP41CV. Accordingly, it is not envisioned that the average Army officer who uses the HP41CV on real world problems which push the calculator to the limits of its capability would write their own programs. In particular, it was never intended that the students in the Combined Arms and Services Staff School would be taught calculator programming. It is a tribute to the power of the device and the quality of the calculator software when a relatively inexperienced user can run complex

programs using little more than the digit entry keys and the run-stop key on the calculator. This does not mean that the user must not have a clear understanding of his problem or the solution technique, but rather it means that the calculator should not require programming skill or extensive training prior to application.

The growing complexity of calculator programs described above and the realization that calculator programs for Army field use are not programmed in the field, suggest the need for a system to support the development, distribution and maintenance of calculator software. An operations research analyst or other professional programmer must be able to more efficiently prepare calculator programs than by keying them into the hand-held device. By preparing the programs initially on a larger computer, such as the IBM 3033, the programmer can use the speed and storage capability of the larger machine to great advantage. In addition, the availability of a full-screen video text editor speeds the process of program revision and maintenance. By providing a capability to integrate comments directly into the source code on the larger computer, program documentation is more easily provided. Essentially the idea is that a programmer would write the calculator program using a terminal

connected to a large computer. After the calculator program is entered into the large computer, a compiler program running on the large computer would check the calculator program for errors and convert the mnemonic instructions into the "key codes" which are the numeric instructions actually executed by the calculator. Then an emulator program running on the large computer would take the numeric instructions from the compiler and execute the program--in effect making the large computer produce the same effects as the calculator only much faster and more efficiently for the programmer. Finally, when the program has been written and tested on the large computer, optical bar code is produced which allows for the economical distribution and use of the program in the field. To encourage the calculator programmer to use the system described, this process should occur in an interactive programming environment in which the user can move from one step to another by issuing simple commands such as those listed and described in Figure 2 and receive help or on line instruction whenever desired. Under this proposed system, the advantages of both the larger computer and the hand-held calculator are used appropriately in a mutually supporting manner. This thesis presents two of the components of this proposed system. First, an IBM EXEC II

program is given which provides an interactive programming environment for users operating under IBM's Conversational Monitor System (CMS.) A short discussion of the design of this program and a complete copy of the source code is contained in Appendix D to this thesis. Secondly, a cross compiler written in IBM standard FORTRAN IV is provided for translating calculator mnemonic instructions into the key codes necessary for use by the emulator and also for the production of optical bar code. The term cross compiler refers to the fact that the program runs on one machine (the larger computer) but compiles programs for another machine (the calculator.) A discussion of the design of this program and a complete copy of the source code is contained in Appendix D. To make the program easier to understand and adapt to new requirements, it is modularized into 24 subroutines and is heavily commented.

To illustrate the use of the system, two of the six example programs originally planned are provided in this thesis. Revised plans now call for the remaining four example programs to be issued at a later date as Naval Postgraduate School technical reports. Because the reasons for the delay constitute some of the most important lessons learned from this thesis research, Chapter 2 documents the process

with a technical discussion of the factors involved. The major conclusions described in Chapter 2 are the need for a prioritized list of criteria with which to evaluate calculator programs and the need for more structure in the programming process. Chapter 2 is technically oriented and assumes the reader is familiar with the concepts of structured programming.

Each of the calculator program examples is described in a separate appendix in which the documentation listed in

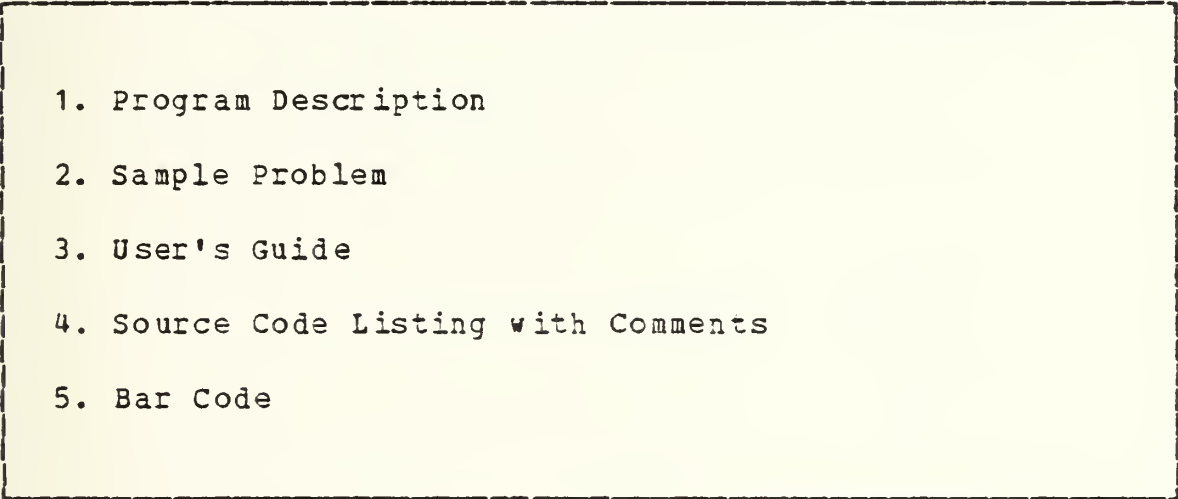
- 
1. Program Description
 2. Sample Problem
 3. User's Guide
 4. Source Code Listing with Comments
 5. Bar Code

Figure 4: Components of Program Documentation

Figure 4 is provided. The first example on single variable statistics is documented in Appendix A and uses the calculator in an area where calculators have long been used, but does so in a way that shows the unique capabilities of the

HP41CV. A second example on linear programming is documented in Appendix B and illustrates an area where calculators have not received widespread application. Most calculator linear programs which have been published to date have been either incomplete algorithms or have been limited to very simple problems.

A third example, which by its nature does not conform to the documentation standards outlined above, describes a set of utility routines which could be distributed in read only memory. Programs for read only memory have different characteristics from other calculator programs and Appendix C is provided to illustrate some of these differences.

II. THE PROGRAMMING ENVIRONMENT

A. CHAPTER OVERVIEW

This Chapter examines calculator programming within the context of the author's experience in preparing HP41CV programs in support of the Combined Arms and Services Staff School. With the advanced capabilities and features of the HP41CV, it was hoped that a complete package of software could be prepared quickly. To document why this did not occur, this chapter will examine strengths and weaknesses of the calculator in relationship to a collection of techniques referred to in computer science as structured programming. For the reader unfamiliar with this term, the previously cited work by Dahl, Dijkstra, and Hoare [Ref. 3] is recommended. This chapter is technically oriented and does assume familiarity with structured programming concepts.

When programming calculator programs for personal use, most programmers, including the author, do not find the task difficult. Programming a hand-held calculator with the capabilities and features of the HP41CV can be a rewarding experience. It is rewarding to master the algorithm of an operations research technique on a hand-held device. The

educational value in programming the calculator has been recognized by many educators, including Hamming [Ref. 4: pp. 2-3] and Weir [Ref. 5: pp. xii-xiii]. Providing a program for general distribution which makes optimum use of the calculator is quite a different situation. It was the author's experience that programs, which gave correct answers when used by the author, often had to be completely re-written several times before being acceptable. This problem became more acute as the size of the programs grew beyond 400 program steps, for at that size it became increasingly difficult to modify programs without affecting the total design. The major conclusions described in this chapter are the need for a prioritized list of criteria with which to evaluate calculator programs and the need for more structure in the programming process.

B. STRUCTURED PROGRAMMING WITH THE HP41CV

1. The Need for Structure

To increase the efficiency of the programming process, a collection of techniques known as structured programming has received widespread attention in the computer science community. While there is no one definition of structured programming, it does require three essential characteristics. First, there must be a logical structure

to the program which reflects the nature of the problem to be solved and any constraints imposed upon the solution. Second, the systematic process of stepwise refinement is used to limit the complexity of program segments. Third, the programming language must reflect the logical structure of the program and assist in stepwise refinement. These three characteristics represent not so much a detailed recipe for program development as they do a philosophy of how programs can be more efficiently written. It was with this philosophy in mind, that a calculator programming support system was proposed which could take into account the strengths and weaknesses of the calculator; balance the structured programming philosophy with the other criteria listed below; and thereby solve the problems encountered in writing calculator software for the Combined Arms and Services Staff School.

2. Fundamental Limitations of Calculators

Writing programs to solve complex problems on a hand-held calculator is difficult both because of inherent limitations in the calculation speed and memory capacity of the machine and also the inability of the calculator's native programming language to directly support structured programming constructs. In many respects, the task is

similar to writing assembly level language programs for larger computers. Calculator programming features a powerful instruction set including advanced mathematical functions but lacks any ability to refer to variables by name instead of storage address. Like assembly language, the calculator's programming language consists of short mnemonic instructions typically followed by the storage location of the data to which the operation is to be applied. While a large amount of computer programming is still done in assembly language, it is generally accepted that programming in a higher level language such as FORTRAN is preferable. Programs written in an assembly language take more time to write and are not as easily changed as higher level language programs. Also, because they depend on the instruction set of a particular machine, they can not be easily transferred from one computer to another. These same disadvantages apply to calculator programming. In addition, because the hand-held device does not have the speed and memory capability of the larger machine, the calculator programmer must be even more mindful of the need to optimize his program to save program steps and execution time.

3. Modular Design

The HP41CV supports structured programming as well or better than any other hand-held calculator. As described in the owner's manual [Ref. 2: pp. 177-196], the machine primitive instruction XEQ encourages the construction of modular programs using calculator subroutines. Each subroutine can be a self-contained unit capable of being written and tested independently and used by multiple programs. This modularity is most strongly encouraged when routines in read only memory are used, for then the application programmer can significantly reduce the number of program steps in his own program. This modularity, however, is not complete, since all variables are globally referenced and can be changed deliberately or inadvertently by any subroutine. This problem is no more apparent than with the use of read only memory, since one of the most limiting factors in using the read only memory programs as subroutines is conflict in the use of common registers. Also, unlike the modularity required in truly structured programs, there is no restriction limiting a subroutine to a single entry and a single exit point. In structured programs, such limits on entry and exit serve to define the fundamental building blocks by which stepwise refinement is made possible. With

the calculator, however, multiple entry and exit points are most useful for allowing a common routine to handle a duplicity of problem conditions. In this thesis, for example, programs are given for which two standard entry points are provided. One entry point uses an alpha-numeric label and an audio prompt to speed data entry, while a second entry point uses the alpha-numeric label but suppresses the audio tone. After data entry, the value entered is displayed, and the user is required to verify the accuracy of the data entered. By using the same subroutine with different entry points, memory space is saved overall at the sacrifice of the structured programming philosophy.

4. Control of Program Flow

A basic deficiency prohibiting the HP41CV from directly supporting structured programming is the way in which program flow is controlled. Programming languages which support structured programming typically have instruction constructs such as WHILE--ENDWHILE, REPEAT--UNTIL, or LOOP--QUIT--ENDLOOP which make programming loops clear and concise. Constructs such as IF--THEN--ELSEIF--ELSE--ENDIF and the CASE statement make the evaluation of conditional expressions efficient and relatively error free. Also, structured programming languages typically discourage the

use of GOTO unconditional transfers because they lead to confusing code. In contrast, the HP41CV programmer must write his own looping constructs and his own conditional evaluation constructs using machine primitive instructions which somewhat obscure the program's basic objective and flow of control. In addition, it is difficult to avoid disturbing pending operations in the stack registers when a conditional statement must be evaluated. As can be seen by the short program shown in Figure 1, the notation of the programming language does not permit structured program flow.

5. Clarification of Program Structure

Because no calculator, including the HP41CV, supports named variables, the use of comments as an integral part of the calculator program is vital if the logical structure of the program is to be made clear as required by structured programming. Comments should provide the variable names when storing and recalling data; they should provide clarification of program flow; and they should mark subroutine boundaries and entry and exit points to make it easier to identify segments of the program. With the HP41CV's stack oriented architecture, it is also frequently useful to display the names of the contents of each of the

Given the number n in the x-register, this program fragment will sum the data values stored in memory locations 1 through n .

<u>Instruction</u>	<u>Comment</u>
LBL "SUM"	To execute press "XEQ SUM".
1E3	
/	
1	
+	Establishes a loop counter.
0	Clears x and pushes loop
LBL 00	counter into y.
RCL IND Y	Recall the next data value.
+	Accumulate the sum.
ISG Y	Increment the loop counter.
GTO 00	If more data remains, branch;
RTN	else, quit and display sum.

Figure 5: Example Program to Add n Numbers

stack registers. In Appendix C on common subroutines with read only memory application, a shell sort [Ref. 6: pp. 84-95] routine is given which employs the technique of using comments to display the names of the variables on the stack register.

6. Data Types and Indirect Addressing

Calculator programs represent more than a sequence of keystrokes; they also represent the manipulation and transformation of data. For maximum efficiency, the manipulation of data should be structured so as to prevent common programming errors. For this reason, most computer languages which directly support structured programming enforce data type correspondence between data and operations. Frequently the formal declaration and initialization of variables is also required. The HP41CV handles two types of data--real numbers and alphanumeric characters. While no formal declaration of variables is required, type checking is done automatically and is transparent to the user. Any attempt to perform an arithmetic operation on alpha-numeric data will result in the message "ALPHA DATA" and the program will halt.

Because there is no formal declaration of variables, the programmer writing programs for the HP41CV must use

extreme caution in managing his data set and insuring that the numbers stored and recalled by the calculator program are in fact the data elements desired. A typical example of an improper data reference occurs when a program is using indirect addressing and attempts to store or recall data from a non-existent data register. This programming error is so common that a special error message "NONEXISTENT" is provided by the calculator when this error is detected. Indirect addressing is an important feature which gives the calculator a considerable amount of power and flexibility, but also represents an additional responsibility for the programmer to explicitly control. On the HP41CV all indirect addressing calculations must be specifically provided by the application program--there are no vector or array data types such as usually found with higher level languages. In an attempt to make indirect addressing more transparent to the programmer, an experimental subroutine was prepared to recall an arbitrary element of a matrix stored as a two dimensional array. This subroutine, which is shown in Figure 2, was used in a simultaneous differential equation combat model and the results evaluated. It accomplished the task, but slowed the execution of the program considerably (resulting in an overhead of 10.5 seconds

of extra execution time for every 100 subroutine calls) and did not significantly improve the size or legibility of the application program. Accordingly, this technique is not recommended and indirect addressing remains a task that must be treated explicitly by the application programmer.

C. ADDITIONAL CRITERIA FOR PROGRAM EVALUATION

Calculator programming in many respects resembles a multi-criteria decision problem. On the surface the criteria for program effectiveness are quite straight forward--the program must yield the correct answer, run quickly, require the fewest possible memory registers and be user friendly. Unfortunately, these objectives often conflict and can not always be simultaneously achieved. In particular, the principles of structured programming are often in conflict with the desire to reduce the size of programs and increase their execution speed. It is also true that the objectives of structured programming concern the process of writing programs, whereas the additional criteria listed concern the final program product itself and are therefore logically considered separately. Attempting to achieve all criteria at once can lead to failure, and some tradeoffs must be considered to evaluate programs and guide program development. The following criteria represent

Entry to this routine assumes the x register contains the column number and the y register contains the row number. The base address must be stored in R04 and the dimension of the matrix must be stored in R05.

```
1 LBL "RCLM
2 RCL 04      (BASE ADDRESS REGISTER
3 +          (ADD BASE TO COLUMN NUMBER
4 X<>Y       (RECALL THE ROW NUMBER
5 1
6 -
7 RCL 05      (DIMENSION OF THE MATRIX
8 *
9 +          (ADDRESS IS NOW IN X REG
10 RCL IND X  (RECALL THE DATA DESIRED
11 RTN
12 END
```

Figure 6: Program to Recall an Element of a Matrix

"lessons learned" in developing application programs as examples for this thesis.

1. User Friendliness

User friendly programs consider the application environment and do not task the user to be all knowing or without error in entering data. While individuals differ greatly with experience, the average user will make frequent errors in entering data with the hand-held calculator's small keyboard. In talking with officers who had used the TI-59 calculator in the field for fire direction, it was discovered that most preferred to use the printer with the calculator because it allowed data to be checked after entry. This was in spite of the fact that the printer and calculator combination is more costly, less portable and less suitable for use in the field than the calculator alone. In short, user friendliness was more important than these other criteria. For this reason, it should be mandatory that any calculator programs intended for Army use in the field must allow the verification of data after entry. Because the use of the printer obviates many of the advantages of the hand-held calculator, the printer should not be required for this verification. One of the considerable advantages of the HP41CV is that the large amount of program

memory makes it possible to store the input values and perform this verification. However, programs written with this criteria in mind may not appear be most efficient to the casual observer.

Another important aspect of user friendliness is limiting the complexity of the calculator and the actions required to get results. The typical Army officer has little appreciation for the multitude of scientific and mathematical functions labeling the keys of the HP41CV. Yet the common programming practice of using the top two rows of calculator keys to indicate the identity of a variable either upon input or output increases confusion over the use of the function keys. This works as follows: When local alphabetic labels are used in a program to represent entry points by which a user indicates the identity of an input variable or requests a particular output variable, then the first two rows of keys on the HP41CV become subroutine execution keys pointing to these local labels when the calculator is in user mode. This feature was very important on the HP67 and TI-59 where the lack of alpha-numeric capability required this method of program execution in order to most easily determine the identity of the input or output value, but it is less important on the HP41CV. It is almost always

true that a program which requires the use of local labels is harder to use, and requires more frequent reference to the user instructions than a program which uses only the run-stop key and properly prompts the user and labels output values.

2. Execution Speed

The second most important criteria for a calculator program is that it must yield results relatively quickly. In preparing example programs for this thesis, this point became very clear when testing two particular programs. One program, a simultaneous differential equation combat model, required in excess of 150 data values in order to yield results. It should be noted that it was only with the introduction of the HP41CV that it became feasible to consider such large problems on a hand-held device. To accommodate the size of the model, the program was written so as to economize on program steps at the expense of increased execution time. It became immediately obvious upon initial testing that this had been the wrong priority--for users of the program were not impressed with either the use of the calculator or the utility of the combat model. If such user acceptance is not present, then the calculator program will remain unused, no matter how elegant the design to conserve

memory. In contrast, the linear programming example given in Appendix B was written so as to emphasize speed even if it meant including code redundancy. This program has been well received in part because it is so much faster than paper and pencil methods.

The easiest and most effective technique that is useful in increasing speed is to decrease the number of program steps that the calculator must process inside program loops. For example, if two different program options require similar but slightly different actions within a program loop, it is tempting to insert a program flag check and branching instructions within a loop so as to use the same loop for both conditions. But this means that the calculator must test the flag and branch inside the loop even though the program is probably shorter overall.¹

Instead, if the application permits, the memory capacity of the HP41CV can be used to best advantage by testing the flag once and then providing separate program loops for the two conditions. Again, this does not appear elegant to the casual observer, but it may result in a more successful program overall. This principle was discovered while

¹ Branching is required when the flag tests either set or clear if more than one instruction is required to account for the differences in the two conditions.

programming the single variable statistics program given in Appendix A. Initially, this program used a common loop for all data input and output operations, including reviewing the input data and making individual corrections. By providing a separate, somewhat redundant loop for data correction, the time required to input data points was reduced.

D. A PROGRAMMING SUPPORT SYSTEM

Considering the structured programming philosophy discussed above in paragraph B and the additional criteria for evaluating programs listed in paragraph C, it becomes immediately obvious that programming with the calculator alone will never meet even a majority of these objectives. It must be recognized that the problem under consideration is not how the average person who owns a calculator should proceed to program it for his own personal use, but rather how the Army can best provide the most cost-effective computational resource for field use. For these reasons, a comprehensive programming support system is required. The programming support system outlined here will consider only the requirement for cost-effective preparation and maintenance of the calculator programs and not the broader issues of distribution and logistic support for the entire

calculator system to include hardware, training materials and printed references.

1. A Cross Compiler and Bar Code Generator

The first requirement for an operational support package is to free the programmer from the limitations of the hand-held calculator itself. Even with the printer and other peripherals, the calculator is no match for the larger machine when large programs must be examined or edited. In addition, the calculator is not currently capable of producing its own optical bar code as required for economic reproduction and distribution of the software. Accordingly, a cross compiler for the HP41CV was listed as the first requirement of the programming support system. Such a cross compiler has been written and is the major outcome of this thesis effort. This cross compiler accepts an HP41CV program written in the language of the calculator and returns the finished bar code as output. Any valid HP41CV program will be processed without need for modification by the cross compiler. In addition to the basic language of the calculator, the user is allowed to inject comments directly into the source code with the use of the left parenthesis as a comment indicator mark. The ability to make comments directly in the source code makes the calculator programs

more legible and more easily modified at a later date or by another programmer. Often, well placed comments can make up for a lack of structure in the program itself as far as legibility and maintainability are concerned. Having the comments directly in the source code facilitates their use and helps insure that they are as up to date as the program. For the average programmer, use of unmodified HP41CV source code augmented with a comment indicator will represent the most common use of the cross compiler. The cross compiler is described in more detail in Appendix D including a complete listing of the source code.

2. A Calculator Emulator

After the calculator source code has been processed by the cross compiler, a need exists to be able to run the program without the wait for the generation of bar code. In addition, for the future development of read only memories for the calculator, an emulator program is required because the calculator itself can store only up to 2000 instructions in active random access memory. The read only memory can store up to four times this amount. Thus, the calculator by itself may not be capable of testing extremely large programs or programs with large amounts of constant data also stored in the read only memory. Although an emulator was

not written for this thesis, the design of the cross compiler reflects the need for such a program. For example, the cross compiler generates an intermediate array of decimal integers which represent the machine language of the HP41CV prior to conversion to binary. It was intended that these decimal integers could be used without modification or further translation within a FORTRAN computed goto statement. Thus, with the difficult translation, instruction parsing and syntax recognition already performed by the cross compiler routines, the emulator could consist of one large FORTRAN loop wherein a decimal integer was addressed in the instruction array by a program pointer variable. The integer is then immediately sent to a computed goto statement which would branch to the appropriate line of FORTRAN code which would simulate the referenced instruction, including updating the stack and the program pointer as appropriate.

3. A Higher Level Language Compiler

The final component in the calculator programming support system would be a program that would translate a higher level language such as PASCAL into HP41CV language which could then be sent to the cross compiler for verification and generation of the bar code and intermediate

calculator language listings. It is the higher level language compiler that would most directly make up for the weakness of the calculator in supporting structured programming. It would be able to increase the modularity of programs, provide for named variables, make indirect addressing transparent and provide structured statements such as WHILE--ENDWHILE and IF--THEN--ELSE. Again, the design of the cross compiler anticipates this requirement and provides a considerable number of subroutines that would also be required by a higher level language compiler. These subroutines include a complete set of string functions for manipulating character data in FORTRAN and an instruction parser. Because it was envisioned that the higher level language compiler would also be able to process statements entered directly as HP41CV instructions, the cross compiler is constructed so that the routine which compiles individual lines of HP41CV source code could be called as a subroutine by the higher level language compiler. Thus, all three major components of the proposed calculator programming support system would work together efficiently.

APPENDIX A

SINGLE VARIABLE STATISTICS EXAMPLE

INTRODUCTION:

Calculating single variable statistics is one of the most frequently used applications of programmable calculators. Army division level staff officers use single variable statistics to summarize and describe data for command briefings and periodic reports. The text by Mendenhall, Scheaffer and Wackerly [Ref. 7: pp. 3-13] is recommended as an introduction to the statistical measures calculated by the program given in this appendix. This program automatically calculates:

- Mean and Median
- Sample Standard Deviation
- Sum of the Squared Deviations about the Mean
- Coefficients of Skewness and Kurtosis
- Minimum, Maximum and Range
- Histogram Cell Frequencies

A single variable statistics program has been given as an example because of its immediate utility to the staff officer and to illustrate several features of the HP41CV which make it a superior device for Army field use. The most important of these features is alphanumeric prompting for input data values. The program given in this appendix provides an alphanumeric prompt for every input and output value and requires only the digit entry keys and run/stop key for data entry. Another important feature of the HP41CV used by this program is its large memory capacity. This program retains up to 219 data points in the calculator's memory to allow the user to review the input data and make corrections during data entry. The large amount of memory allows the calculator to sort the data and calculate the order statistics including the minimum, maximum and median. Calculation of the median is a feature of this program which distinguishes it from other calculator statistics programs. In addition, without having to re-enter the data, the histogram may be calculated with a varying number of cells or a varying cell width.

PROGRAM DESCRIPTION:

The single variable statistics program has entry points for two different techniques of data input. The fastest

method, which provides both an alphanumeric prompt and an audio tone to speed data entry, may be called by execution of the program from entry point "STAT1." A slower method, which provides greater accuracy and suppresses the audio tone for classroom use, may be called by execution of the program from entry point "S1." When called from "S1," the program requires the verification of each data point after entry. The sequence of actions is as follows:

1. The calculator displays an alphanumeric prompt. As an example, "X1?" is the prompt for the first point.
2. The user enters the data value with the digit entry keys and presses the run/stop key.
3. The calculator displays the data entered with a label derived from the alphanumeric prompt. For example, "x1=3.1415" is a typical calculator response. This display is prompting the user to verify the correctness of the data displayed.
4. If the value is correct, then the user simply presses the run/stop key and the calculator advances to the next point.
5. If the value is erroneous, the user enters the correct value with the digit entry keys and then presses the run/stop key. Then the calculator will again repeat step 3 and ask the user to verify the data value. This process will continue until the user makes no modification to the data value.

To run the program from either entry point the user may use the XEQ key, or assign the entry point label ("STAT1" or "S1") to a key and execute it by pressing that key in the USER mode. Further instructions on running programs and making key assignments are contained in the calculator owner's manual [Ref. 2: pp 114-116].

In addition to the two initial entry points described above, several other alphabetic labels provide the user with functions that are called outside the normal sequence of program execution. Label "SR" provides the user with the capability to review the data stored in calculator memory, either before or after the data has been sorted. When used before the sort, the "SR" function is most useful in verifying the entire data set at one time. If used for this purpose, it should be called after all of the data has been entered and the mean of the data set is displayed with the "XBAR" label. If flag 21, the printer enable flag, is set "on" during this data review, then the calculator will stop as each point is displayed and the user may make corrections in the same manner as described above for the point-by-point verification associated with the "S1" entry point. When used after the sort, the "SR" function is most useful for displaying the order statistics for the data set. If used for this purpose, it should be called after the histogram is output--when the "CMD" prompt is displayed. If the user presses run/stop after the "CMD" prompt, the order statistics will automatically be displayed.

The design of the program, especially the data entry loop, reflects the need for calculation speed. Code

redundancy exists at several points in order to reduce the need for extra flags, labels and goto statements which would slow execution during data entry. In spite of this need for speed, the summary totals needed for calculation of mean, standard deviation, skewness and kurtosis are accumulated during data entry. This is done so that these summary statistics are available with little or no wait following data entry.

A complete listing of the program registers and flags used by this program is shown at the end of the program listing.

SAMPLE PROBLEM:

In order to establish a training standard for an obstacle course, a division assistant G3 randomly selects 10 soldiers and records the time it takes each to complete the course. The following times in minutes were recorded:

2.1	2.4	2.2	2.7	2.5
2.4	2.6	2.6	2.3	2.9

Determine the summary statistics and cell frequencies necessary to plot a histogram of this data.

SOLUTION:

1. First, set the size of the calculator's data memory large enough to retain the data values. This requires at least 16 registers plus 1 for each data point, or a total of 26 in this example. Alternatively, the size of data memory may be set arbitrarily large, up to a maximum of 235 provided the user has no other programs in the calculator he wishes to retain. For this example press:

```
XEQ  ALPHA  SIZE  ALPHA  26
```

2. To call the program, determine the appropriate method of data entry and select the corresponding entry point.

Press:

```
XEQ  ALPHA  STAT1  ALPHA  (quick entry)
      -OR-
XEQ  ALPHA  S1     ALPHA  (classroom use)
```

3. The calculator will respond with the prompt "N?" asking for the number of data points. Press:

```
10  R/S
```

4. The calculator will respond with the prompt "X1?" asking for the first data point. Press:

```
2.1  R/S
```


5. If you called the program via "S1" the calculator will respond with "X1=2.100" asking for verification that the first point is correct. If not correct enter the correct value, else press run/stop.

6. The calculator will continue in the same way as steps 4 and 5 for the remaining data points until all the data has been entered. If at any time you discover that you have made an error in data entry for any point, press:

XEQ ALPHA SC ALPHA

The calculator will respond with the prompt "POINT?" asking for the number of the point in error. For example, if point number 5 were in error, you would then press:

5 R/S

Assuming you had just input a 5 as the point in error, the calculator would then respond with the prompt "X5?" asking for the correct value of point 5. Respond with the correct value and press run/stop. The calculator will then go back to the place in the data entry sequence where it left off or it will go to the calculation of the summary statistics if data entry was previously completed.

7. When data entry has been completed, the calculator will respond with the mean of the data sample labeled as follows:

XBAR=2.470

At this point, you have the option of reviewing the entire data set or continuing to calculate the remainder of the statistics. To review the entire data set, press:

XEQ ALPHA SR ALPHA

Note that if flag 21 is set on (press SF 21), the calculator will stop after each data point is displayed, permitting you to change any value simply by entering the new value and pressing run/stop.

8. After the mean is displayed with the "XBAR" label, if you simply press the run/stop key, the calculator will calculate the following statistics with the label shown: After each press R/S.

<u>Display</u>	<u>Meaning</u>
SSQD=0.521	Sum of Squared Deviations About the Mean
SX=0.241	Sample Standard Deviation
SKEW=0.170	Skewness
KURTO=2.302	Kurtosis

9. At this point the calculator will automatically sort the data. This may take from several seconds to several minutes

depending on the number of points in the data set. After the data set has been sorted, the calculator will display the median as follows:

MED=2.400 TO	(Press R/S)
.. 2.500	

Two data values are displayed because when the number of data points is even, the median is not unique, but rather spans an interval from the one point listed above to the other. Many users may wish to simply take the middle of this interval as the median, but any point is technically correct in the interval. When the number of data points is odd, the median is unique and only one value will be displayed by the calculator.

10. After the median is displayed as described in step 9, the calculator will display the following statistics labeled as shown:

<u>Display</u>	<u>Meaning</u>
MIN=2.100	Minimum Value
MAX=2.900	Maximum value
RNG=0.800	Range

11. At this point the calculator will respond with "CELL?" asking for the number of cells the user desires in the

histogram. If the number of cells is not significant at this point, the calculator will pick an appropriate number if the user simply presses run/stop. For this example, press:

R/S

12. Next the calculator responds with "WIDTH" asking for the width of the cells. Simply press run/stop if you do not wish to establish the width manually. Again, you may see the width the calculator will use by pressing the clear arrow key (Unless the width is an integer, you will also need to press FIX 3 to display the decimal properly if you wish to examine the width.) For this example, press:

R/S

13. The calculator will now display the cell frequency counts as an integer count followed by the next cell boundary. The leftmost cell boundary is set equal to the minimum value and is not explicitly output. If a data point falls exactly on a cell boundary, it is counted in the left cell.

For this example, the display will show:

<u>Display</u>	<u>Meaning</u>
CNT=2	Two observations
xx=2.260	between 2.1 (the minimum) and 2.26 (the cell boundary)
CNT=3	Three observations
xx=2.420	between 2.26 (see above) and 2.42 (the next boundary)
CNT=1	One observation
xx=2.580	between 2.42 (see above) and 2.58 (the next boundary)
CNT=3	Three observations
xx=2.740	between 2.58 (see above) and 2.74 (the next boundary)
CNT=1	One observation
xx=2.900	between 2.74 (see above) and 2.90 (the maximum)

14. After the last cell boundary is displayed, the calculator will display "CMD" asking the user for the next command. Frequently, the user will wish to modify the histogram by changing the number of cells or the cell width. To recalculate the histogram cell frequencies without re-entering the data press:

XEQ ALPHA AGAIN ALPHA

If no further work with the histogram is desired, the user may view the order statistics simply by pressing run/stop.

USER INSTRUCTIONS:

SINGLE VARIABLE STATISTICS

STEP	EXPLANATION	SEE	PRESS	RESULT
1	SET SIZE (nnn=16+NUMBER OF DATA POINTS)		XEQ "SIZE NNN	UP TO nnn = 235
2	CALL THE PROGRAM ("STAT1 IS FOR REGULAR USE) ("S1 IS FOR CLASSROOM USE)		XEQ "STAT1 -or- "S1	
3	ENTER THE NUMBER DATA POINTS.	N?	input R/S	
4	ENTER THE DATA For mistakes or to review the data see last two steps below. WHEN VERIFY MODE IS SET ON (SET BY FLAG 05 ON), AFTER EACH DATA POINT IS ENTERED, THE VALUE WILL BE ECHOED BACK BY THE CALCULATOR.	X1?, X2? ETC. x1=xx etc.	input R/S R/S -or- correct value	
5	SUMMARY STATISTICS ARE CALCULATED WHEN ALL DATA HAS BEEN ENTERED. STANDARD DEVIATION SKEWNESS KURTOSIS	XBAR=xx SSQD=xx SX=xx SKEW=xx KURT=xx	R/S R/S R/S R/S R/S	mean sum of sq dev from mean
6	CALCULATOR WILL AUTOMATICALLY SORT DATA POINTS. AND THEN DISPLAY: MEDIAN (note if N is even the median is not unique and an inter- val is displayed) MINIMUM MAXIMUM RANGE	PRGM MED=xx MIN=xx MAX=xx RNG=xx	 R/S R/S R/S R/S	STANDBY

USER INSTRUCTIONS:

SINGLE VARIABLE STATISTICS

STEP	EXPLANATION	SEE	PRESS	RESULT
7	USER OPTION TO ENTER NUMBER OF HISTOGRAM CELLS. NO INPUT IS REQUIRED.	CELL?	R/S -or- INPUT N R/S	
8	USER OPTION TO ENTER WIDTH OF HISTOGRAM CELLS. (HAS PRECEDENCE OVER NUMBER OF CELLS IF A WIDTH IS ENTERED.)	WIDTH?	R/S -OR- INPUT R/S	
9	CALCULATE HISTOGRAM (OUTPUT DATA ABOUT EACH CELL FROM LEFT TO RIGHT.)	CNT=II	R/S	CELL FREQ COUNT
		XX=xx	R/S	UPPER X-VALUE LIMIT
10	ACCEPT NEXT COMMAND	CMD ~	ENTER NEXT CMD	
11	RECALCULATE HISTOGRAM		XEQ "AGAIN	
12	EDIT AN INPUT VALUE AT ANY TIME PRIOR TO DATA SORT. AFTER INPUT OF NEW VALUE CALCULATOR WILL RETURN TO DATA INPUT OR CALCULATION OF SUMMARY STATS AS APPROPRIATE.	POINT?	XEQ "SC INPUT POINT NUMBR	WILL REMOVE POINT
		X?	INPUT CORRECT VALUE	

USER INSTRUCTIONS:

SINGLE VARIABLE STATISTICS

STEP	EXPLANATION	SEE	PRESS	RESULT
13	REVIEW DATA POINTS (OR REVIEW ORDER STATS AFTER SORT.)		XEQ "SR	


```

      (-----)
      |          |
      |   STAT1   |
      |          |
      |-----|
1  LBL "STAT1      (RECOMMENDED ENTRY POINT
2  CF 05          {SET VERIFY MODE OFF
3  SF 26          {ENABLE AUDIO
4  GTO "SS
      |-----|
      |          |
      |   S1      |
      |          |
      |-----|
5  LBL "S1        (ENTRY POINT FOR CLASSROOM USE
6  SF 05          {SET VERIFY MODE ON
7  CF 26          {DISABLE AUDIO TONES
8  SF 21          {SET TO STOP DURING VERIFICATION
      |-----|
      |          |
      |   "SS     |
      |          |
      |-----|
9  LBL "SS        (ENTRY POINT FOR USER SET OPTIONS
10 CF 29          {NO DIGIT GROUPING
11 &REG 10        {ESTABLISH STATISTICAL REGISTERS
12 CF 06          {USED BY DATA REVIEW FUNCTION
13 CF 08          {USED BY DATA EDITING FUNCTION
      |-----|
      |          |
      |   00      |
      |          |
      |-----|
                                DATA ENTRY (F06-CLEAR)
                                AND
                                DATA REVIEW (F06-SET)
14 LBL 00
15 15
16 STO 04          (ESTABLISH INDIRECT ADDRESS BASE REG.
17 STO 00          {INITIALIZE DATA ENTRY POINTER
18 RCL 15          {NUMBER DATA POINTS (LAST PROBLEM)
19 CLX
20 "N?
21 FC? 06          (CLEAR MEANS DATA ENTRY, NOT REVIEW
22 PROMPT
23 1E3
24 /
25 1
26 +
27 STO 01          (SET UP LOOP COUNTER FOR DATA POINTS

```



```

      { (-----)
      {                                } DATA ENTRY LOOP
      {                                }
      {-----}
28 LBL 01
29 ISG 00      (INCREMENT DATA STORAGE POINTER
30 LBL 02
31 RCL IND 00  (RECALL DATA VALUE
32 "X
33 FIX 0
34 ARCL 01
35 FIX 3
36 ASTO 03     (TEMP STORAGE FOR LABEL
37 FS? 06     (IS THIS REVIEW OF DATA PREV. ENTERED?
38 GTO 03
39 "┐=?
40 TONE 9      (PROMPT USER FOR NEXT DATA VALUE
41 PROMPT
42 STO IND 00  (STORE THE DATA VALUE
43 FC? 05     (NO VERIFICATION OF DATA DESIRED?
44 GTO 04
45 LBL 03     (FOLLOWING IS THE VERIFICATION ROUTINE
46 CLA
47 ARCL 03     (RECALL THE LABEL
48 "┐=?
49 ARCL IND 00 (RECALL THE STORED DATA
50 CF 22      (CLEAR DATA ENTRY FLAG
51 AVIEW
52 FC? 22     (WILL STOP FOR DATA ENTRY IF F21 SET
53 GTO 04     (WAS THERE NO DATA CHANGE DURING VIEW?
54 STO IND 00 (IF THERE WAS A NEW VALUE, THEN RECORD
55 GTO 03     (IT AND GOBACK AND RE-VERIFY THE DATA.
56 LBL 04     (FOLLOWING IS THE STATISTICAL ACCUM.
57 ST+ 10     (STORES SIGMA X
58 X|2
59 ST+ 11     (STORES SIGMA X-SQUARED
60 LASTX
61 *
62 ST+ 12     (STORES SIGMA X-CUBED
63 LASTX
64 *
65 ST+ 13     (STORES SIGMA X-FOURTH-POWER
66 FS? 08     (IS THIS A DATA REVIEW?
67 RTN
68 ISG 01     (IF DATA ENTRY, INCREMENT INPUT CNTR.
69 GTO 01
70 RCL 01     (AT END OF DATA ENTRY, RECALL INPUT
      {      (COUNTER, WHICH IS A NUMBER EQUAL ONE
      {      (MORE THAN NUMBR POINTS

```


SINGLE VARIABLE STATISTICS

60

HP41C SOURCE CODE:

SINGLE VARIABLE STATISTICS

```

125 *
126 4
127 *
128 -
129 RCL 03      (XBAR
130 X|2
131 RCL 11      (SIGMA X-SQUARED
132 *
133 6
134 *
135 +
136 RCL 15      (NUMBER POINTS
137 /
138 RCL 03      (XBAR
139 4
140 Y|X
141 3
142 *
143 -
144 STO 07      (FOURTH MOMENT
145 RCL 05      (SECOND MOMENT
146 X|2
147 /
148 "KURT=
149 XEQ 97      (OUTPUT THE KURTOSIS
              {PS? 09
              {RTN
              (SHORT FORM WOULD NOT COMPUTE STATS
              (WHICH REQUIRE SORTED DATA
150 XEQ 98      (CALL A DATA SORTING ROUTINE
151 CF 00       (INITIALIZE TEMP FLAG USED TO CHECK
152 RCL 15      (EVEN OR ODD NUMBER OF DATA POINTS
153 2
154 /
155 FRC
156 X=0?        (WAS IT AN EVEN NUMBER OF POINTS?
157 SF 00       (IF WAS EVEN NUMBER, SET FLAG.
158 LASTX
159 .5
160 +           (COMPUTING ADDRESS OF MEDIAN
161 RCL 04      (ADDRESS BASE REGISTER
162 +           (X-REG NOW HAS ADDRESS OF MEDIAN
163 "MED=
164 ARCL IND X
165 FC? 00      (EVEN NUMBER POINTS IMPLIES THE MEDIAN
166 GTO 05      (NOT UNIQUE,BUT SPANS AN INTERVAL
167 " TO
168 PROMPT     (DISPLAY THE LEFT BOUNDARY OF MEDIAN
169 1
170 +           (X-REG POINTS TO RIGHT BOUND OF MEDIAN
171 ". ."
172 ARCL IND X
173 LBL 05
174 PROMPT

```



```

      { (-----)
      { "AGAIN                                     DISPLAY HISTOGRAM
      { -----}

175 LBL "AGAIN
176 RCL 04                                     (ADDRESS BASE REGISTER
177 1
178 +
179 RCL IND X                               (RECALL THE FIRST ORDER STAT
180 "MIN                                     (CALL AN OUTPUT LABELING ROUTINE
181 XEQ 97                                   (HOLDS STARTING (LEFTMOST) X BOUNDARY
182 STO 09                                   (ADDRESS BASE REGISTER
183 RCL 04                                   (NUMBER OF DATA POINTS
184 RCL 15
185 +
186 RCL IND X                               (RECALL THE N-TH ORDER STATISTIC
187 "MAX                                     (DISPLAY THE MAX VALUE OBSERVED
188 XEQ 97                                   (MIN
189 RCL Z
190 -
191 STO 08                                   (TEMP STORE FOR THE RANGE
192 "RNG                                     (DISPLAY THE RANGE
193 XEQ 97                                   (INITIALIZE TEMP FLAG TO MARK LAST BAR
194 CF 00                                   (NUMBER DATA POINTS
195 RCL 15                                   (ADDRESS BASE REGISTER
196 RCL 04
197 +
198 1E3
199 /                                         (COMPUTING INDEX LOOP COUNTER
200 RCL 04                                   (ADDRESS BASE REGISTER
201 +
202 1
203 +
204 STO 01                                   (R01 SET TO ADDRESS AND LOOP THRU DATA
205 RCL 15                                   (NUMBER POINTS
206 LN
207 2
208 *                                         (DEFAULT NUMBER OF BARS IS 2*LN(N)
209 FIX 0                                   (VALUE IS ROUNDED NOT TRUNCATED
210 RND
211 "CELL?                                   (USER HAS OPTION TO CHNGE NUMBR CELLS
212 PROMPT                                   (RANGE
213 RCL 08
214 X<>Y
215 /
216 "WIDTH                                   (USER HAS OPTION TO CHANGE CELL WIDTH
217 PROMPT                                   (NOW HOLDS CELL WIDTH NOT RANGE
218 STO 08
219 LBL 06
220 RCL 08                                   (CELL WIDTH
221 ST+ 09                                   (UPPER LIMIT OF CURRENT CELL COUNTED
222 0
223 STO 02                                   (INITIALIZE CELL COUNTER
224 LBL 07
225 RCL IND 01                               (NEXT DATA POINT
226 RCL 09                                   (CELL UPPER LIMIT
227 FIX 3
228 RND

```


HP41C SOURCE CODE:

SINGLE VARIABLE STATISTICS

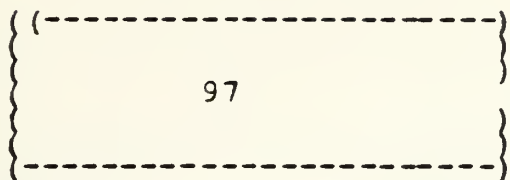
229	X<Y?	(DATA POINT LESS THAN UPPER LIMIT
230	GTO 08	
231	1	
232	ST+ 02	(INCREMENT THE CELL COUNTER
233	ISG 01	{PREPARE TO LOOK AT NEXT DATA POINT
234	GTO 07	
235	SF 00	(SET FLAG FOR OUTPUT OF LAST BAR
236	LBL 08	
237	"CNT	
238	RCL 02	
239	FIX 0	
240	XEQ 97	(OUTPUT THE CELL FREQUENCY COUNT
241	FIX 3	
242	"XX	
243	RCL 09	
244	XEQ 97	(OUTPUT CELL BOUNDARY--LOWER LIMIT
245	FC?C 00	{IS THIS THE LAST BAR ?
246	GTO 06	
247	"CMD	
248	AVIEW	
249	RTN	

SINGLE VARIABLE STATISTICS

64

HP41C SOURCE CODE:

SINGLE VARIABLE STATISTICS



OUTPUT LABELING ROUTINE

```
287 LBL 97
288 "┐=
289 ARCL X
290 PROMPT
291 RTN
```


SINGLE VARIABLE STATISTICS

66

THIS PROGRAM USES THE FOLLOWING REGISTERS:

R00 -- INPUT DATA ADDRESS POINTER
 R01 -- LOOP INDEX COUNTER
 (USED BY DATA ENTRY AND SORT ROUTINES)
 R02 -- TEMP REGISTER
 {CELL FREQUENCY COUNT IN HISTO RTN}
 {AND SHELL BOUNDARY IN SORT ROUTINE)
 R03 -- TEMP REGISTER
 {INPUT LABEL IN DATA INPUT ROUTINE)
 {XBAR IN SUMMARY STAT ROUTINE)
 {AND POINTER IN SORT ROUTINE)
 R04 -- INDIRECT ADDRESS BASE
 R05 -- SECOND MOMENT (POPULATION VARIANCE)
 R06 -- THIRD MOMENT
 R07 -- FOURTH MOMENT
 R08 -- HISTOGRAM CELL WIDTH
 R09 -- TEMP REGISTER
 {HOLDS SUM OF SQ DEVIATION ABOUT MEAN)
 {AND HISTOGRAM CELL UPPER LIMIT)
 R10 -- SUM OF X VALUES
 R11 -- SUM OF X-SQUARED VALUES
 R12 -- SUM OF X-CUBED VALUES
 R13 -- SUM OF X RAISED TO THE FOURTH POWER
 R14 -- NOT USED BUT SET TO ZERO BY CLR&
 R15 -- NUMBER DATA POINTS (SET BY &+)
 R16....R228 RAW DATA POINTS
 -- IN NATURAL SEQUENCE BEFORE SORT
 -- AS ORDER STATISTICS AFTER SORT

THIS PROGRAM USES THE FOLLOWING FLAGS:

F00 -- TEMP FLAG (USED IN EDIT AND HISTO RTNS)
 F05 -- VERIFY MODE (EVERY DATA POINT ECHOED)
 F06 -- INDICATES REVIEW OF DATA NOT QUERY MODE
 F08 -- INDICATES EDITING A DATA POINT
 F21 -- PRINTER ENABLE (STOPS CALCULATOR
 DURING AVIEW INSTRUCTION)
 F26 -- AUDIO ENABLE

340 END

S I N G L E V A R I A B L E S T A T I S T I C S

1



2



3



4



5



6



7



9



9



10



11



12



S I N G L E V A R I A B L E S T A T I S T I C S

13



14



15



16



17



18



19



20



21



22



23



24



S I N G L E V A R I A B L E S T A T I S T I C S

25



26



27



28



29



30



31



32



33



34



35



36



S I N G L E V A R I A B L E S T A T I S T I C S

37



38



39



40



41



42



43



44



45



APPENDIX B

LINEAR PROGRAMMING EXAMPLE

INTRODUCTION:

Linear programming is an operations research technique normally associated with computerized data bases and the largest computers. Because of the complexity of the computer programs for linear programming and the large amount of data associated with most real world problems, calculators have not been widely used for this application. With the increased memory capacity of the HP41CV, however, it is now possible to offer a calculator program which can solve interesting small scale linear programs. Of value primarily as an educational aid, this program will also be able to solve many small scale problems found at Army division, brigade and battalion level. The text by Hillier and Lieberman [Ref. 8: pp. 16-66] is recommended as an introduction to the theory of linear programming as used by the program given in this appendix. Use of the program requires the user to formulate the linear programming problem; set up a Simplex tableau in standard form including adding slack, surplus and artificial variables as required; and interpret

the final tableau including the calculation of the values associated with the variables in the final basis. Using the tableau form of the Simplex algorithm, the calculator performs both phase I (to obtain a feasible solution) and phase II (to obtain an optimal solution) to solve the linear programming problem. The calculator automatically determines the pivot column and pivot row for each pivot step. Infeasible and unbounded problems are automatically identified for the user by the program. There is no explicit handling of variables with upper bounds.

PROGRAM DESCRIPTION:

The program is written as a series of subroutines, each of which performs a major step in the Simplex algorithm. To provide clarity to the user, alphabetic labels have been retained to identify the subroutines in lieu of faster and more memory efficient numeric labels. The alphabetic labels have not been retained for use as program entry points and may be changed to numeric labels at the option of the user. The program has two entry points, "LP" for running a new problems and "ALP" for reviewing data previously entered.

Subroutine "FINDQ" determines the pivot column by selecting the variable in the objective function with the most negative "price." If "FINDQ" discovers at least one

negative value in the objective function, then the tableau column number associated with the most negative value will be stored in register 05. Upon return from "FINDQ," the main routine tests register 05 to see if it contains a non-zero entry. If the entry is zero, it means that no further pivots will improve the value of the objective function, and the Simplex algorithm halts. If the program was in phase I (flag 11 clear) and the value of the phase I objective function is reduced to zero, then a feasible solution has been found and the program will automatically proceed to phase II to discover an optimal solution.

Subroutine "FINDP" determines which variable will leave the current basis by performing a minimum positive ratio test along the pivot column. In this way, the pivot row is determined. The row number of the pivot row is stored in register 06. Upon return from subroutine "FINDP," the main routine tests register 06 to see if it contains a non-zero entry. If the entry is zero, it means that the problem is unbounded and the Simplex algorithm halts. Such an unbounded condition is most likely caused by an error in the problem formulation.

Having determined the pivot column and the pivot row, subroutine "PIVOT" performs the actual Simplex pivot

operation. To speed calculation register 00 is used as a temporary register to hold the reciprocal of the pivot element. Note that the pivot row is handled separately from the other rows in the tableau. Flag 04 is used to provide the option of stopping calculation after every pivot. When this flag is set, the program will halt to allow the user to review the status of the tableau with the "ALP" function.

Subroutine "CHECK" has two primary functions. First, it is used to verify that the designated basic variables are in row elimination form prior to the start of the Simplex algorithm. This means that the basic variable must have a coefficient of 1 in the row in which it is basic and zero's in all other rows. The second function of check is to prepare the objective function for phase I, if the initial basis contains artificial variables as indicated by one or more minus signs in the "JB" vector.

Three subroutines are used to query the user for input data. Subroutine "READMN" queries the user for the number of constraints and decision variables in the problem and verifies the calculator memory is set to contain all the data necessary to solve the problem. Subroutine "READJB" queries the user for a column vector of pointers which indicate which variable is currently basic in each row. When

entering this vector of pointers, the user indicates artificial variables with a minus sign. Subroutine "READA" queries the user for the values in the initial Simplex tableau including the slack and surplus variables and the right hand side and objective function.

Several other service routines also are provided in this program. Memory size verification is done by subroutine "SIZE," which is called from within "READMN." Subroutine "IN" is used to query the user for data entry and is called by all of the data input routines. Subroutine "NXT" initializes registers which contain frequently used quantities such as the the total size of the tableau for phase I and phase II. Subroutine "INIT" clears the calculator memory and sets flags and program constants appropriately for input of a new problem. Subroutine "SETL" establishes the loop counters used repeatedly within almost every other subroutine. Subroutine "ERR1" displays an appropriate error message if a data entry error is detected.

SAMPLE PROBLEM:

A division assistant G4 is planning an ammunition upload plan. There are four types of tank munitions to consider,

including:

A = Discarding Sabot Rounds
B = High Explosive Anti-Tank Rounds
C = Phosphorous Munitions
D = Machine Gun Ammunition

Based on the Commander's guidance the assistant G4 is to consider the sabot rounds as twice as important as the HEAT rounds, which in turn are themselves twice as important as a unit amount of phosphorous munitions and machine gun ammunition. His mission then, is to maximize:

$$Z = 4A + 2B + C + D$$

He is, however, constrained by the following factors:

1. There can be no more than 30 units of both sabot and HEAT munitions combined.
2. There can be no more than 50 units of all types of ammunition combined.
3. There must be at least 30 units of HEAT and phosphorous munitions combined.
4. There must be at least 5 units of machine gun ammunition.

These constraints may be expressed as:

$$\begin{array}{rcl} A + B & \leq & 30 \\ A + B + C + D & \leq & 50 \\ B + C & \geq & 30 \\ D & \geq & 5 \end{array}$$

Based on the Commander's guidance and the constraints listed above, formulate an optimum load plan. Fractional units are permitted.

SOLUTION:

1. Before beginning with the calculator, the first step is to layout the tableau in standard form. This step and the last step of interpreting the final tableau require working knowledge of linear programming as explained in Hillier and Liberman [Ref. 8: pp. 16-66]. The standard form of the tableau is:

1 A	2 B	3 C	4 D	5 H1	6 H2	7 S1	8 S2	9 A1	10 A2	11 RHS
1	1					1				30
1	1	1	1				1			50
	1	1		-1				1		30
			1		-1				1	5
-4	-2	-1	-1							0

In this tableau, H1 and H2 are surplus variables; S1 and S2 are slack variables; and A1 and A2 are artificial variables.

2. The first step with the calculator is to set the size of the calculator's data memory. This program requires 20 registers for temporary storage, 1 register for each tableau element, and 1 register for each row to hold the pointer to the basic variable for that row. Thus, if M is the number of constraints and N is the number of variables including slack, surplus and artificial variables, then the total data

storage requirement is:

$$\text{storage required} = 21 + M + ((N + 1) \times (M + 2))$$

As mentioned in the program description, the "SIZE" subroutine will automatically verify that the user has allocated enough data storage to solve the problem. The length of the program is such that 177 data storage registers is the maximum number of data storage registers that can be allocated. Thus, linear programs with 7 constraints and 15 decision variables can be solved with this program. For this example, press:

```
~      XEQ      ALPHA      SIZE      ALPHA      175
```

3. Call for execution of the program with a new data set.

Press:

```
      XEQ      ALPHA      LP      ALPHA
```

4. The calculator will respond with the prompt "NUM ROWS?" asking for the number of constraints in the linear program formulation. In this example, press:

```
4  R/S
```

5. The calculator will respond with the prompt "NUM COLS?" asking for the number of variables in the problem. The user

must count the number of slack, surplus and artificial variables in this total. In this example, press:

10 R/S

6. The calculator will respond with the prompt "BASIC 1 ?" asking for the variable number of the variable which is basic in the first row. One of the major features of this program is that the basic variables need not be in the rightmost positions in the tableau. Thus, if a tableau were given in which some pivots had already been performed, the program could resume operation immediately. In this example, press:

7 R/S

In a similar fashion, the calculator will then query the user for the variable number of the variables which are basic in the remaining rows.

For this example:

<u>See</u>	<u>Respond With</u>
Basic 2 ?	8 R/S
Basic 3 ?	9 CHS R/S
Basic 4 ?	10 CHS R/S

Notice that because the basic variables in rows three and four are artificial variables, the variable number is entered as a negative number. This signals the calculator

that these variables must be driven from the basis in order to reach an initial feasible solution.

7. Next, the calculator will respond with "T1,1?" asking for the first element in the tableau. The user should enter the numbers in the tableau using the digit entry keys and pressing run/stop after every entry. Notice that the right hand side and the objective function will be entered with the appropriate index in the tableau as shown in step 1 above. The user must insure that the objective function is in standard form with the appropriate sign for each coefficient--in this example each coefficient is negative.

8. After the last element in the tableau has been entered, the calculator will begin to automatically perform the Simplex algorithm. If the user wishes to stop the calculator after every pivot, he may at any time press:

R/S SF 04 R/S

If this flag is set, the calculator will stop and display the pivot number after every pivot is completed.

9. When the Simplex algorithm can no longer improve the objective function, the calculator will stop and display the value of the objective function. In this example, the

calculator will stop after approximately three minutes and display:

VALUE=110.000

10. At this point, the user must use entry point "ALP" to determine the status of the final tableau. For this example, press:

XEQ ALPHA ALP ALPHA

Then by sequentially pressing the run/stop and clear arrow keys, the basic variables and tableau entries will be displayed. For example, in this problem:

<u>See</u>	<u>Press</u>	<u>See</u>	<u>Meaning</u>
BASIC 1?	CLX	2	Variable 2 is basic in the first row.
BASIC 2?	CLX	1	Variable 1 is basic in the second row.
etc.			

Then for the elements of the tableau:

<u>See</u>	<u>Press</u>	<u>See</u>	<u>Meaning</u>
T1,1?	CLX	0.000	Tableau entry
T1,2?	CLX	1.000	Tableau entry

11. After the calculator is finished, it remains for the user to interpret the final tableau. Again, the reference by Hillier and Lieberman [Ref. 8: pp. 16-66] is of primary value. In particular, the user must be able to determine the value of the final decision variables based upon what

variables are in the basis, and what the final "right hand side" values are for each row. For this example, the final tableau is:

1 A	2 B	3 C	4 D	5 H1	6 H2	7 S1	8 S2	9 A1	10 A2	11 RHS
	1			-1	-1	1	-1	1	1	15
1				1	1		1	-1	-1	15
		1			1		1		-1	15
			1		-1				1	5
-----0-----0-----0-----0-----2-----2-----1-----3-----2-----2-----110-----										

Thus, the solution may be interpreted as 15 units each for munitions A,B and C and 5 units for munition D.

USER INSTRUCTIONS:

LINEAR PROGRAMMING

STEP	EXPLANATION	SEE	PRESS	RESULT
1	SET SIZE ($NNN = 21 + M + (N + 1)(M + 2)$) WHERE M=NUM ROWS AND N=NUM COLS		XEQ "SIZE NNN	UP TO N IN = 177
2	CALL THE PROGRAM		XEQ "LP	
3	ENTER THE NUMBER OF CCNSTRANTS	NUM ROW ?	INPUT M R/S	
4	ENTER THE NUMBER OF VARIABLES (INCLUDE SLACKS, SURPLUS & ARTIF.	NUM COL ?	INPUT N R/S	
5	ENTER CURRENT BASIC VARIABLE NUMBERS BY ROW	BASIC 1 ? ETC.	INPUT VAR # R/S	
6	ENTER TABLEAU VALUES. FOR MISTAKES OR TO REVIEW THE DATA SEE LAST STEP BELCW.	T1,1? ETC.	INPUT R/S	
7	TO FORCE THE CAL- CULATOR TO STOP AFTER EACH PIVOT.		R/S SF 04 R/S	
8	SIMPLEX COMPLETED: OPTIMAL SOLUTION FOUND.	VALUE= XX.XXX		FINAL OBJ. FUNCTION VALUE
9	SIMPLEX COMPLETED: PROBLEM IS INFEASIBLE.	INFEAS		

LINEAR PROGRAMMING

85

86

HP41C SOURCE CODE:

LINEAR PROGRAMMING

```
55 ARCL 05
56 STOP
57 LBL 40
58 XEQ "PIVOT
59 GTO 15
```



```
( (-----)
      FINDQ
(-----)
```

```
60 LBL "FINDQ
61 SF 01
62 0
63 STO 03
64 STO 05
65 1
66 RCL 11
67 XEQ "SETL
68 LBL 31
69 RCL 14
70 RCL 01
71 +
72 STO 00
73 RCL IND X
74 RCL 03
75 -
76 RCL 07
77 CHS
78 X<Y?
79 GTO 38
80 FC? 11
81 GTO 37
82 RCL 15
83 RCL 01
84 +
85 RCL IND X
86 ABS
87 RCL 07
88 X<=Y?
89 GTO 38
90 LBL 37
91 RCL IND 00
92 STO 03
93 RCL 01
94 INT
95 STO 05
96 LBL 38
97 ISG 01
98 GTO 31
99 CF 01
100 RTN
```



```
      (-----)
      {          }
      {  FINDP  }
      {          }
      (-----)

101 LBL "FINDP
102 SF 02
103 0
104 STO 06
105 1E20
106 STO 03
107 1
108 RCL 08
109 XEQ "SETL
110 LBL 41
111 RCL 01
112 1
113 -
114 RCL 12
115 *
116 RCL 04
117 +
118 STO 00
119 RCL 05
120 +
121 RCL IND X
122 STO 02
123 RCL 07
124 X>Y?
125 GTO 48
126 RCL 00
127 RCL 12
128 +
129 RCL IND X
130 RCL 02
131 /
132 STO 00
133 RCL 03
134 -
135 RCL 07
136 CHS
137 X<Y?
138 GTO 48
139 LBL 47
140 RCL 00
141 STO 03
142 RCL 01
143 INT
144 STO 06
145 LBL 48
146 ISG 01
147 GTO 41
148 CF 02
149 RTN
```



```

      (-----)
      |          |
      |   PIVOT   |
      |          |
      (-----)

```

```

150 LBL "PIVOT
151 SF 03
152 RCL 06
153 1
154 -
155 RCL 12
156 *
157 RCL 04
158 +
159 STO 03
160 RCL 05
161 +
162 RCL IND X
163 1/X
164 STO 00
165 1
166 RCL 12
167 XEQ "SETL
168 LBL 51
169 RCL 03
170 RCL 01
171 +
172 RCL 00
173 ST* IND Y
174 ISG 01
175 GTO 51
176 1
177 RCL 09
178 FS? 11
179 0
180 FC? 11
181 1
182 +
183 XEQ "SETL
184 LBL 52
185 RCL 06
186 RCL 01
187 INT
188 X=Y?
189 GTO 59
190 1
191 -
192 RCL 12
193 *
194 RCL 04
195 +
196 STO 16
197 RCL 05
198 +
199 RCL IND X
200 STO 00
201 2
202 RCL 12
203 XEQ "SETL

```


HP41C SOURCE CODE:

LINEAR PROGRAMMING

```
204 LBL 53
205 CF 07
206 RCL 05
207 RCL 02
208 INT
209 X=Y?
210 SF 07
211 RCL 03
212 +
213 RCL IND X
214 RCL 00
215 *
216 RCL 16
217 RCL 02
218 +
219 X<>Y
220 ST- IND Y
221 FC? 07
222 GTO 54
223 0
224 STO IND Z
225 LBL 54
226 ISG 02
227 GTO 53
228 LBL 59
229 ISG 01
230 GTO 52
231 RCL 13
232 RCL 06
233 +
234 RCL 05
235 STO IND Y
236 1
237 ST+ 17
238 CF 03
239 TONE 7
240 FC? 04
241 RTN
242 "PIVOT "
243 ARCL 17
244 PROMPT
245 RTN
```



```
(-----)
{          }
{ READMN  }
{          }
(-----)
```

```
246 LBL "READMN
247 7
248 STO 00
249 "NUM ROWS
250 XEQ "IN
251 XEQ "NXT
252 XEQ "NXT
253 "NUM COLS
254 XEQ "IN
255 XEQ "NXT
256 RCL 10
257 *
258 RCL 04
259 +
260 1
261 ST+ 00
262 RDN
263 STO IND 00
264 RCL 09
265 +
266 XEQ "SIZE?
267 FC?C 25
268 PROMPT
269 RTN
```



```
      { (-----)
      {
      {           SIZE?
      {
      {-----}
270 LBL "SIZE?
271 "SIZE>=
272 ARCL X
273 1
274 -
275 SF 25
276 RCL IND X
277 RTN
      {-----}
      {
      {           "NXT
      {
      {-----}
278 LBL "NXT
279 1
280 ST+ 00
281 +
282 STO IND 00
283 RTN
      {-----}
      {
      {           SETL
      {
      {-----}
284 LBL "SETL
285 1E03
286 /
287 1
288 +
289 STO IND Y
290 RTN
```



```
      (-----)
      {          }
      { READJB   }
      {          }
      (-----)
291 LBL "READJB
292 1
293 RCL 08
294 XEQ "SETL
295 RCL 13
296 STO 00
297 FIX 0
298 LBL 01
299 "BASIC "
300 ARCL 01
301 " "
302 XEQ "IN
303 ISG 01
304 GTO 01
305 FIX 4
306 RTN
```



```

      (-----)
      |         |
      |   READA   |
      |         |
      (-----)
307  LBL "READA
308  SF 10
309  1
310  RCL 09
311  XEQ "SETL
312  LBL 11
313  2
314  RCL 12
315  XEQ "SETL
316  LBL 12
317  FIX 0
318  "T
319  ARCL 01
320  "1
321  ARCL 02
322  FIX 4
323  LBL 13
324  RCL 01
325  1
326  -
327  RCL 12
328  *
329  RCL 02
330  +
331  RCL 04
332  +
333  1
334  -
335  STO 00
336  LBL 14
337  FC? 10
338  GTO 16
339  XEQ "IN
340  GTO 17
341  LBL 16
342  "1=
343  ARCL IND 00
344  AVIEW
345  LBL 17
346  ISG 02
347  GTO 12
348  ISG 01
349  GTO 11
350  CF 10
351  0
352  RTN

```


LINEAR PROGRAMMING

4


```

      (-----)
      {          }
      {    CHECK    }
      {-----}

374 LBL "CHECK
375 SF 11
376 1
377 RCL 12
378 XEQ "SETL
379 RCL 04
380 RCL 09
381 RCL 12
382 *
383 +
384 STO 14
385 STO 15
386 LBL 91
387 RCL 14
388 RCL 01
389 +
390 0
391 STO IND Y
392 ISG 01
393 GTO 91
394 1
395 RCL 08
396 XEQ "SETL
397 LBL 92
398 CF 07
399 RCL 13
400 RCL 01
401 +
402 RCL IND X
403 X<0?
404 SF 07
405 ABS
406 STO 00
407 X=0?
408 GTO "ERR1
409 RCL 12
410 X<=Y?
411 GTO "ERR1
412 2
413 RCL 09
414 XEQ "SETL
415 LBL 93
416 CF 08
417 RCL 01
418 INT
419 RCL 02
420 INT
421 X=Y?
422 SF 08
423 1
424 -
425 RCL 12
426 *
427 RCL 00

```


HP41C SOURCE CODE:

LINEAR PROGRAMMING

```
428 +  
429 RCL 04  
430 +  
431 FC? 08  
432 0  
433 FS? 08  
434 1  
435 STO IND Y  
436 ISG 02  
437 GTO 93  
438 FC? 07  
439 GTO 98  
440 CF 11  
441 RCL 14  
442 RCL 00  
443 +  
444 1  
445 STO IND Y  
446 2  
447 RCL 12  
448 XEQ "SETL  
449 LBL 96  
450 RCL 01  
451 INT  
452 1  
453 -  
454 RCL 12  
455 *  
456 RCL 02  
457 +  
458 RCL 04  
459 +  
460 RCL IND X  
461 CHS  
462 RCL 14  
463 RCL 02  
464 +  
465 X<>Y  
466 RCL IND Y  
467 +  
468 STO IND Y  
469 ISG 02  
470 GTO 96  
471 LBL 98  
472 ISG 01  
473 GTO 92  
474 FC? 11  
475 RTN  
476 RCL 04  
477 RCL 08  
478 RCL 12  
479 *  
480 +  
481 STO 14  
482 RTN
```


THE FOLLOWING TABLE DESCRIBES THE KEY REGISTER AND FLAG ASSIGNMENTS MADE BY THIS PROGRAM:

R00 = TEMPORARY REGISTER. HOLDS RECIPROCAL OF PIVOT
 ELEMENT IN SUBROUTINE PIVOT.
 R01 = LOOP INDEX COUNTER
 R02 = LOOP INDEX COUNTER
 R03 = TEMPORARY REGISTER. HOLDS MIN VALUE IN FINDQ;
 MAX VALUE IN FINDP; AND IS AN INTERMEDIATE
 ADDRESS CALCULATION VALUE IN PIVOT.
 R04 = BASE REGISTER FOR INDIRECT ADDRESSING
 (SET = 19)
 R05 = THE PIVOT COLUMN NUMBER
 R06 = THE PIVOT ROW NUMBER
 R07 = EFFECTIVE ZERO LEVEL
 R08 = M = NUMBER OF ROWS
 R09 = M PLUS 1
 R10 = M PLUS 2
 R11 = N = NUMBER OF VARIABLES
 R12 = N PLUS 1
 R13 = BASE REGISTER FOR THE LOCATION OF THE
 VECTOR JB WHICH CONTAINS POINTERS TO
 WHICH VARIABLE IS BASIC IN EACH ROW.
 R14 = ROW NUMBER OF THE OBJECTIVE FUNCTION;
 SET TO M PLUS 1 OR M PLUS 2 AS DETERMINED
 BY NEED FOR PHASE I.
 R15 = BASE REGISTER FOR THE LOCATION OF THE
 PHASE I OBJECTIVE FUNCTION, IF NEEDED.
 R16 = TEMPORARY REGISTER.
 R17 = NUMBER OF PIVOTS PERFORMED.
 R18 = RESERVED FOR FUTURE USE.
 R19-R177 = DATA STORAGE REGISTERS FOR ELEMENTS OF
 THE TABLEAU AND THE JB VECTOR.

FLAGS

F01 - F03 = SUBROUTINE EXECUTION FLAGS. BECAUSE THESE
 FLAGS ARE VISIBLE IN THE DISPLAY THEY CAN
 BE SET WHEN ENTERING A MAJOR SUBROUTINE AND
 CLEARED WHEN LEAVING -- GIVING THE USER A
 VISUAL INDICATION OF WHAT IS HAPPENING
 INSIDE THE CALCULATOR.

F01--SUBROUTINE FINDQ
 F02--SUBROUTINE FINDP
 F03--SUBROUTINE PIVOT

F04 = WHEN SET, STOPS CALCULATOR AFTER EACH PIVOT.
 F07 = USED AS TEMPORARY FLAG IN PIVOT AND CHECK ROUTINES.
 F10 = USED AS A TEMPORARY FLAG IN READ ROUTINES.
 F11 = WHEN SET, INDICATES PHASE II IS IN PROGRESS.
 F29 = CONTROLS FORMAT OF DISPLAY SEPARATOR.

483 END

"LP" LINEAR PROGRAMMING

1



2



3



4



5



6



7



8



9



10



11



12



"LP" LINEAR PROGRAMMING

13



14



15



16



17



18



19



20



21



22



23



24



" L P " L I N E A R P R O G R A M M I N G

25



26



27



28



29



30



31



32



33



34



35



36



"LP" LINEAR PROGRAMMING

37



38



39



40



41



42



43



44



45



46



47



48



"LP" LINEAR PROGRAMMING

49



50



51



52



53



54



55



56



57



58



59



60



61



62



63



64



65



66



67



68



69



70



71



72



"LP" LINEAR PROGRAMMING

73



74



75



76



77



APPENDIX C

SUBROUTINES FOR READ ONLY MEMORY

INTRODUCTION:

The calculator subroutines described in this appendix perform functions which are frequently required by application programs and are therefore ideal candidates for use in a read only memory (ROM.) These routines were written especially to illustrate the differences between read only memory routines and routines designed for individual use via bar code or magnetic cards. These differences include more attention to entry and exit point options, an attempt to keep the size of the routines as compact as possible, and an attempt not to disturb the register stack if at all possible.

These common subroutines are provided separately from application programs because when more than one application program uses the routines, as is recommended for these programs, the use of a separate block of common functions saves space in the ROM overall. Also, by providing a convenient set of "macro" instructions, application programs can be constructed more quickly and easily. Because these

subroutines are used frequently, they have been individually optimized and tested to save memory space and execution time. By using these "macros" within an application program, the application programmer can be reasonably certain of their efficiency and reliability.

Almost all user/calculator interface is handled by these routines. There is one set of subroutines which assumes the user has a printer, and one set which does not. Printer instructions are preceeded in the listings shown in the appendix by an (PRT: label. When not using these routines on read only memory, the user will load one set or the other (but not both), as appropriate for his/her application. In so doing, the user with the printer gets full benefit from it while the user without the printer pays no penalty in execution time or memory space for the calculator's print instructions. Also, to change from use of the printer to use of the calculator without it, the user needs only to read in the new common block--the application programs are retained in memory unchanged. The subroutines appear the same to any application program--giving the added benefit that any application program which uses them for input or output operations will automatically make good use of

the printer even if written by a programmer who did not explicitly consider a printer requirement.

When using these common subroutines, a discipline is enforced upon the application program concerning use of the calculator memory registers. This saves the programmer from having to plan his "register map" from scratch for each new program. Also, it insures compatability across different application programs for similar data objects such as matrices and loop index counters. One of the most annoying problems with read only memory programs available from the calculator manufacturer is this lack of cross program compatability. Conflict in the use of memory registers is the rule, rather than the exception; and it is frequently impossible to efficiently use more than one read only memory program as a subroutine within a user written program. A third reason why register assignment standards are advantageous is that they make it easier for the user of the calculator to remember the key register assignments and, if necessary, recall their contents during the execution of an application program.

Another function performed by this set of common subroutines is to simplify the use of indirect addressing--a critical goal on the HP41CV.

Because the common subroutines listed in this appendix are always called by application programs and never from the keyboard by the user, the typical user instructions are inappropriate. Instead, for the benefit of application programmers wishing to use the routines, the basic functions and structure of each are explained in subsequent sections of this appendix.

Subroutine "IN"

Subroutine "IN" is used as a general input and output interface between the user and the calculator. This subroutine has three alternative entry points which when called affect functions as follows:

```
IN--Input mode (displays a question mark query)
IO--Output mode (displays labeled data value)
IX--Direct mode (input of value in x register)
```

In particular, one entry point, "IN", may be called whenever an application program must query the user for a numeric input value. As such, it is a direct replacement for the PROMPT instruction organic to the calculator, but automatically prompts, verifies and stores the received value using an indirect address contained in register 00. The printer version of the subroutine will automatically label and print the final, verified data value recorded.

Subroutine "IN" uses register 00 as a data location pointer and automatically increments this register so that subsequent calls to the subroutine will result in sequential data manipulation. The application programmer must insure that register 00 contains a number equal to the storage register number prior to calling the subroutine. For example, if register 00 contains 17, "IN" will store the data in register 17. One of the major advantages of this routine is that the same subroutine may be used to verify and/or change the data previously recorded. Thus, separate edit routines are usually unnecessary. Pressing the R/S key without touching any other key leaves the value stored unchanged. Pressing "1" and "+" and then "R/S" adds one to the current stored value, and so on for other function keys. Entering a new string of digits results in that new value being stored.

An additional feature of this subroutine is the "verify" mode indicated by flag 05. Flag 05 is reserved for this purpose and is set "on" by a call to subroutine "VR"--another of the subroutines in this common set. The verify mode is intended for use by a novice or other user who wishes to verify every data value as it is keyed into the calculator. The advantage is increased accuracy and confidence in the result.

Subroutine "D2"

Subroutine "D2" is used to set up the index register for a program loop. This subroutine has two alternative entry points which when called, increment different index registers as follows:

D2--Establishes register 02 as the index
D1--Establishes register 01 as the index

This subroutine is intended for use with the "ISG" loop structure which has the effect most like that of a FORTRAN "DO LOOP." For example, to execute a loop 20 times:

```
...  
20  
XEQ "D1  
LBL 00  
....  
ISG 01  
GTO 00  
...
```

The advantage of this form of loop structure is that register 00 may be used within the loop for address calculations. The first time the loop is addressed the integer portion of the number in register 00 will be 1, the second time it will be 2 and so on. There is no need to truncate the fractional portion of the number because the HP41C ignores the fractional component of a number when calculating a register address. Use of index registers for address calculation makes indirect addressing practical.

Registers 01 and 02 should be reserved for use as index registers by the application programmer. In most cases these two registers should prove sufficient.

Subroutine "VR"

Subroutine "VR" is used as a general purpose calculator initialization routine. This subroutine has three alternative entry points which vary the amount of initialization performed as follows:

- VR--Sets the verify mode on, and the following:
- WR--Suppresses the audio tones, and the following:
- WS--Clears all memory,
 - Sets the display for integers,
 - Assigns statistical registers,
 - Sets "zero" level for equality testing, and
 - Sets base address for indirect addressing.

In the printer version of this initialization routine, the subroutine prints a banner (usually the program name) which has been stored in the alpha register prior to calling the initialization subroutine.

If flag 13 is set prior to calling the initialization routine, then the calling program must have placed the number of data registers required to execute the program in the x register prior to calling the initialization routine. In this case, a check will automatically be performed using subroutine "SZ" described below.

It is recommended that all application programs provide an alternate entry point which bypasses the initialization

step. Then if a data error is encountered, the user may review the data entered into the calculator by simply pressing the return key once after every prompt. This procedure works because subroutine "IN" recalls the stored value prior to prompting the user. When the user presses the clear key, the alphabetic prompt is removed and the existing data value revealed.

Subroutine "SZ"

Subroutine "SZ" is used to test if sufficient numbers of data registers are available to run an application program. This subroutine may be either called directly or as part of the initialization routine described above.

Subroutine "ER"

Subroutine "ER" is called whenever the application program encounters an error--usually in the input data. A prompt is displayed and an audio tone sounded, provided flag 26 (the audio enable flag) has not been cleared by the initialization routine described in paragraph D.

Subroutine "SORT"

This subroutine is included to illustrate the use of a stack register table in the program comments, but it also represents a useful utility routine. The sorting algorithm

used is the shell sort [Ref. 6: pp. 84-95] which gives reasonably fast sorting times with a very small program size. All conventions such as base register in R04 and number of data points in R15 are assumed by this subroutine. A complete list of all such register assignments is listed at the end of the program listing.

Subroutines "PUT" and "GET"

These two small routines provide a useful capability to store and recall up to three integers between 0 and 999 in one data register. This means that if you are manipulating a spread sheet of small, positive integers you can store the same data in one third the space. Of course, run time is degraded (about 20 seconds for every 100 data references.) To store a value, assuming the base register has been defined, just press:

```
value ENTER| point-number XEQ "PUT
```

To recall a value, simply press:

```
point-number XEQ "GET
```



```

      ( (-----) )
      (           )
      (           )
      (           )
      (           )
      (           )
      (-----) )
1  LBL "IN          (INPUT MODE--DISPLAY LABEL AND ?
2  SF 10           (SET QUERY ONCE FLAG
3  GTO 05
4  LBL "IC          (OUTPUT MODE--DISPLAY LABEL AND DATA
5  CF 10           (INSURE NO QUERY
6  LBL 05
7  RCL IND 00      (ASSUMES R00 POINTS TO STORAGE REG
8  LBL "IX          (DIRECT MODE--ASSUMES X REG HOLDS DATA
9  ASTC 05         (ASSUMES LABEL SET UP BY CALLING PROG
10 LBL 01
11 "=-
12 FS? 10          (QUERY OR DISPLAY VALUE?
13 "=?
14 FC? 10
15 ARCL IND 00     (DISPLAY DATA VALUE
16 CF 22           (DIGIT ENTRY DETECTION FLAG
      (PRT: FS? 10
      (PRT: CF 21
      (DISABLE PRINTER FOR QUERY
17 PRCPMT
      (PRT: SF 21
18 STC IND 00      (STORE INPUT VALUE
19 CLA             (PREPARE ALPHA REG FOR NEXT PROMPT
20 FC? 05          (NOT VERIFY MODE?
21 GTC 03
      (FOLLOWING IS ACTION TAKEN WHEN IN VERIFY MODE
22 FS? 22          (ANY INPUT DETECTED?
23 GTO 02          (REDISPLAY VALUE IF USER INPUT
24 FC? 10          (DISPLAY VALUE ONCE IF NO INPUT
25 GTC 04
26 LBL 02          (DISPLAY LABEL AND DATA VALUE
27 CF 10           (NOTE MUST HAVE QUERIED ONCE
28 ARCL 05
29 GTO 01
      (FOLLOWING IS ACTION TAKEN FOR NON-VERIFY MODE
30 LBL 03
31 FS?C 10         (INPUT MODE?
32 GTO 04          (IF INPUT MODE, EXIT ROUTINE
33 FS? 22          (WAS THERE INPUT ?
34 GTC 02          (USER CHANGED VALUE, SO VERIFY.
35 LBL 04          (PREPARE TO EXIT
      (PRT: FS? 55
      (PRT: GTO 06
      (PRT: LBL 05
      (PRINTER ATTACHED?
36 ISG 00          (INCREMENT POINTER FOR NEXT IC OPERATION
37 RTN
38 RTN            (FINAL VALUE IS IN X REG UPON EXIT
      (PRINT THE FINAL VALUE
      (PRT: LBL 06
      (PRT: CLA
      (PRT: ARCL 05
      (PRT: "=-
      (PRT: ARCL IND 00
      (PRT: PRA
      (PRT: GTO 05

```



```

      ( (-----)
      (
      (
      (          D2
      (
      (-----)
39 LBL "D2          (SETUP LOOP USING REG 2 AS INDEX COUNTER
40 2
41 GTC 07
42 LBL "D1          (SETUP LOOP USING REG 1 AS INDEX COUNTER
43 1
44 LBL 07
45 X<>Y            (NUMBER LOOP ITERATIONS MUST BE IN X
46 IE3             (PRIOR TO CALLING THIS SUBROUTINE.
47 /
48 1
49 +
50 STC IND Y
51 RTN

```



```

( (-----)
(
(          VR          )
(-----)
)

52 LBL "VR          (SET VERIFY MODE ON
53 SF C5          (INITIALIZE FOR CLASSROOM USE -- NO AUDIO
54 LBL "WR          (STANDARD INITIALIZATION ROUTINE FOLLOWS
55 CF 26
56 LBL "WS
57 CLRG
58 &REG 10
59 CF 29          (SETS DISPLAY--NO DECIMAL POINT IF INTEGER
60 IE-C4          (EFFECTIVE ZERO LEVEL
61 STC 03
62 15
63 STO 04          (NORMAL INDIRECT ADDRESS BASE REGISTER.
64 1              (THIS IS 1 LESS THAN THE NUMBER OF THE
65 +
66 STO 00          (FIRST REGISTER WHERE DATA IS STORED.
    (PRT: ADV
    (PRT: SF 12      (DOUBLE WIDE PRINTING
    (PRT: FC? 55     (PRINTER ATTACHED?
    (PRT: PRA
    (PRT: CF 12      (SET BACK TO SINGLE WIDE PRINT
    (PRT: ADV
67 FC? 13          (HAS INITIALIZATION ROUTINE BEEN ASKED TO
68 RTN              (VERIFY SIZE?
69 X<> Z

```



```

      ( (-----)
      (
      (
      (          SZ
      (
      (-----)
70  LBL "SZ
71  "SET SZ=
72  ARCL X
73  1
74  -
75  SF 25          (PREPARE TO IGNORE ERROR
76  RCL IND X      (TEST FLAG 25 TO SEE IF SUFFICIENT
77  FC?C 25
78  PRCPMT
79  RTN            (SIZE EXISTS.
      (-----)
      (
      (          ER
      (
      (-----)
80  LBL "ER        (DISPLAY "DATA ERROR" PROMPT & SOUND TCNE.
81  0
82  LN            (BEST WAY TO DETERMINE WHERE ERROR
83  TONE 2         (OCCURED IS TO HIT THE SST KEY ONCE,
84  RTN           (THEN GO INTO PRGM MODE.

```


COMMON SUBROUTINES

120

FP41C SOURCE CODE:

COMMON SUBROUTINES

```
( (-----)
(                                     )
(                                     )
(          SC                        )
(                                     )
(-----)
```

```
133 LBL "SC
134 RCL 00
135 1
136 +
137 STC 05
138 -1EC3
139 STC 10
140 -1
141 3
142 /
143 STC 09
```



```

      ( (-----)
      (
      (          PUT
      (
      (-----)
144  LBL "PUT
145  "VALUE
146  11
147  STC 00
148  XEQ "IA
149  SF C2
      (-----)
      (
      (          GET
      (
      (-----)
150  LBL "GET
151  "BIT REG
152  PRCPMT

```



```

( (-----)
(
(
(      SA
(
(-----)
)

```

```

153 LBL "SA
154 3
155 /
156 INT
157 STC 07
158 LASTX
159 -
160 RCL 09
161 /
162 12
163 +
164 STO C9
165 RCL 05
166 ST+ 07
167 RCL IND 07
168 INT
169 STC 14
170 LASTX
171 -
172 RCL 10
173 *
174 INT
175 STO 13
176 LASTX
177 -
178 RCL 10
179 *
180 INT
181 STC 12
182 RCL 11
183 RCL IND 08
184 STO 11
185 FC?C 02
186 RTN
187 X<>Y
188 STC IND 08
189 RCL 12
190 RCL 10
191 CHS
192 /
193 RCL 13
194 +
195 RCL 10
196 CHS
197 /
198 RCL 14
199 +
200 STO IND 07
201 RTN

```



```
(
(
( THE FOLLOWING TABLE DESCRIBES THE KEY REGISTER AND FLAG
( ASSIGNMENTS MADE BY THIS PROGRAM:
(
```

```
( R00 = INDIRECT ADDRESS FOR STORAGE OF INPUT DATA
( R01 = LOCP INDEX CCOUNTER
( R02 = LOCP INDEX CCOUNTER
( R03 = EFFECTIVE ZERO LEVEL -- USE AS TEMP IF NA
( R04 = BASE REGISTER FOR INDIRECT ADDRESSING (15)
( R05 = TEMP REGISTER FOR ALPHA PROMPT
( R06 - R09 = APPLICATION PROGRAM TEMP REGISTERS
( R10 - R15 = STATISTICAL REGISTERS--USE AS TEMP IF NA
( R16.... = STORAGE OF DATA VIA INDIRECT ADDRESSING
(
```

FLAGS

```
( F00-F04 = SUBROUTINE EXECUTION FLAGS. BECAUSE THESE
( FLAGS ARE VISIBLE IN THE DISPLAY THEY CAN
( BE SET WHEN ENTERING A MAJOR SUBROUTINE AND
( CLEARED WHEN LEAVING -- GIVING THE USER A
( VISUAL INDICATION OF WHAT IS HAPPENING
( INSIDE THE CALCULATOR. USE AS TEMPORARY
( FLAGS IF THIS IS NOT REQUIRED.
(
( F05 = VERIFY INPUT MODE. "ON" WHEN SET. WHEN SET
( AFTER EVERY DATA VALUE IS ENTERED, THE CALC.
( WILL ECHO THE PROMPT AND DATA VALUE ENTERED.
( IF VALUE IS CORRECT, SIMPLY PRESS R/S KEY,
( OTHERWISE ENTER A CORRECTED VALUE AND
( CALCULATOR WILL AGAIN ASK FOR VERIFICATION.
(
( F10 = USED AS A TEMPORARY FLAG INSIDE "IC". INDICATES
( NO QUERY PROMPT IS DESIRED.
( F11 = AUTOMATIC EXECUTION FLAG -- DON'T USE EVER
( F12 = DOUBLE WIDE PRINTING -- LOCAL WITHIN "IO".
( F13 = WHEN SET MEANS MAIN ROUTINE ASKING "VR" FOR
( AUTOMATIC SIZE CHECK AFTER INITIALIZATION.
( APPLICATION PROGRAM MAY USE FREELY AFTER INIT.
(
( F14 - F20 = AVAILABLE FOR APPLICATION PROGRAM USE.
(
```

```
202 END
```


COMMON SUBROUTINES

1



2



3



4



5



6



7



8



9



10



11



12



COMMON SUBROUTINES

13



14



15



16



17



18



19



20



21



22



23



24



COMMON SUBROUTINES

25



26



27



28



29



APPENDIX D

THE CROSS COMPILER PROGRAM AND COMMAND PROCESSOR

DESIGN METHODOLOGY:

This appendix discusses the design methodology used during construction of both the cross compiler program and the command processor, which is an IBM EXEC II program which provides an interactive programming environment for users of the system.

Blazie's compiler for the HP65 calculator [Ref. 9] represents one of the first attempts to provide a compiler for calculator programs. Both Carvalho [Ref. 10: pp. 25-29] and McNeal [Ref. 11: pp. 148-178] have published BASIC language programs which cross compile HP41CV instructions on a microcomputer for output to a line printer which can print acceptable bar code. While these referenced programs provided valuable insights into the problem, especially into the special characteristics of the HP41CV instruction set, none was exactly suited to the needs of this study. Because the Versatec plotter at the Naval Postgraduate School could be easily used only by FORTRAN programs, FORTRAN seemed the computer language of choice for this project. Both programs

were written with limited objectives and neither would have easily supported the extensions desired. Extensions planned for implementation included:

- An extended instruction set.
- In code comments.
- Extensive error checking.
- Compatability with the Emulator.
- Synthetic Instructions [Ref. 1].
- Instruction macro's.

Having decided to code an original cross compiler, a design methodology which would capitalize on the advantages of FORTRAN was planned. FORTRAN's major deficiency for use in constructing a compiler of any type is its lack of alpha-numeric string handling capabilities. Rather than struggle with the lack of string functions, it was decided to code the necessary string functions as separate subroutines. This decision was reinforced countless times throughout the process of writing the compiler. The string function subroutines have been used not only in the cross compiler, but in many other FORTRAN programs since they were originally written. In fact, many persons who have no

interest in the HP41CV cross compiler may find the set of string functions listed in this thesis to be a valuable set of utility routines to be used to augment FORTRAN. The general convention used throughout the string function subroutines is that an alphabetic string may be represented as a vector of two byte integer variables used to store the characters and a single four byte integer variable used to store the length of the string.

One of the major advantages of the cross compiler is its ability to handle comments integrated within the HP41CV source code. This feature is critical to the clarification of the logical structure of the HP41CV programs. Because the parenthesis is not a valid HP41CV character, it was chosen as the comment indicator character. A comment may occur beginning at the first column on an input line or anywhere after an HP41CV instruction. The comment must follow the instruction because everything after the comment mark out to the end of the input line is considered part of the comment.

The control the user has over the output listing is also one of the advantages of the cross compiler. When two comment indicator marks are placed in positions one and two of the input line, the compiler will force a page eject when printing the output listing. In addition, the user can vary

the number of output lines per page and cause useful banners to be placed adjacent to program labels for ease of recognition.

Altogether there are twenty-four subroutines and a main program which constitute the cross compiler. The source code for each of these routines is provided in the second section of this appendix. Each subroutine begins with a statement concerning its function and construction. Accordingly, no general description of each subroutine will be repeated here. However, subroutine COMP deserves special attention, for it is the master lexicographic analyzer for the compiler and would also interface the user with the emulator. Its function is to receive a single line of HP41CV source code and identify it. COMP considers all HP41CV instructions to be of one of three types. The first category are the single byte instructions with no operands that can be compiled by a simple table look up. COMP has been constructed so that the instruction set can be extended at any time simply by increasing the size of this table. In this way abbreviated or altered command names could be easily used. The second category of instructions are the multi-byte instructions which require a table lookup and the translation of one or more operands, including possibly an

indirect instruction indicator. The table examined by the compiler is the same as for the category one instructions, and a code is given in the table which indicates to the compiler the number of operands which are required with each instruction. A syntax check is then made in subroutines IONE and ITWO to insure that the number and characteristics of each operand are appropriate for the given instruction. One of the major advantages of the use of the cross compiler is the syntax and error checking that is performed during the compilation process. The third type of instruction represents the exceptional instructions that are so difficult to compile that they require separate subroutines for efficient compilation. These instructions include storage and recall of data, program labels and program flow control statements such as goto and execute.

In order to provide an efficient programming command system for the compiler that would minimize the need to know technical details about the operation of the compiler, an IBM EXEC II program was written. This program not only interfaces the user to the compiler, but it also provides on line user instructions as to how to use the system. Included in this command processor is a command menu which gives the format and short description for each command.

Another command, help, provides more detailed information about each command. When a novice user first enters the exec, or types the name of the exec followed by a question mark, then he receives a four page narrative description of what the system is, how it works, and what actions he must take to write a successful HP41CV application program.


```

CP SET PF01 IMMED PF13
CP SET PF02 IMMED PF14
CP SET PF03 IMMED PF15
CP SET PF04 IMMED PF16
CP SET PF05 IMMED PF17
CP SET PF06 IMMED PF18
CP SET PF07 IMMED PF19
CP SET PF08 IMMED PF20
CP SET PF09 IMMED PF21
CP SET PF10 IMMED PF22
CP SET PF11 IMMED PF23
CP SET PF12 IMMED PF24
CP SET PF13 IMMED PF13
CP SET PF14 IMMED PF14
CP SET PF15 IMMED PF15
CP SET PF16 IMMED PF16
CP SET PF17 IMMED PF17
CP SET PF18 IMMED PF18
CP SET PF19 IMMED PF19
CP SET PF20 IMMED PF20
CP SET PF21 IMMED PF21
CP SET PF22 IMMED PF22
CP SET PF23 IMMED PF23
CP SET PF24 IMMED PF24
CP TERMINAL LINEND OFF
CP SPJOL PRINTER CONT
CP GLOBAL TXLIB NONIMSL CMSLIB FORTMOD2 MCD2EEH IMSLSP
STATE INSTR CODES A
&IF &RETCODE ^= 0 &USERMODE = R
&IF &INDEX GT 0 &GOTO -PROGNAM
CLRSCRN
&BEGTYPE -ENDINTRO
*** HP41C CROSS COMPILER ***

```

THIS PROGRAM IS USED TO MAKE IT EASIER TO WRITE, DOCUMENT AND REVISE PROGRAMS FOR THE HP41C CALCULATOR. AS OUTPUT, THIS PROGRAM WILL PRODUCE OPTICAL BAR CODE FOR DIRECT ENTRY OF YOUR PROGRAM INTO AN HP41C OR HP41CV CALCULATOR.

WARNING THIS PROGRAM IS PART OF AN ONGOING RESEARCH PROJECT AND AS SUCH IS SUBJECT TO CONSTANT REVISION. WHILE THERE ARE NO KNOWN ERRORS, THE PROGRAM HAS NOT BEEN EXTENSIVELY TESTED. TO INSURE THAT ANY ERRORS YOU DETECT ARE PROMPTLY CORRECTED, IT IS IMPORTANT THAT YOU SUBMIT AN ERROR REPORT TO THE PROGRAM PROPONENT AS SOON AS POSSIBLE.

IN ORDER TO GO FROM A PROGRAM IN YOUR HEAD TO THE FINISHED BAR CODE THERE ARE THREE MAIN STEPS:

- (1) EDIT. THE PROGRAM MUST BE PREPARED AS INPUT TO THE CROSS COMPILER. THE EASIEST WAY TO DO THIS IS WITH THE CMS XEDIT FACILITY.
- (2) COMPILE. THE PROGRAM MUST BE PROCESSED BY THE CROSS-COMPILER. THE CROSS-COMPILER IS ACTUALLY A FORTRAN PROGRAM WHICH PRODUCES TWO CMS FILES AS OUTPUT. BOTH THESE FILES HAVE THE SAME NAME AS YOUR PROGRAM NAME, BUT HAVE DIFFERENT FILE TYPES. THE "LISTING" FILE SHOWS THE RESULTS OF THE COMPILE STEP INCLUDING ANY ERRORS, AND THE "DATA" FILE IS A FILE OF ZERO'S AND ONE'S USED BY THE BAR CODE GENERATOR.
- (3) BAR. THE "DATA" FILE FROM THE COMPILE STEP IS USED AS INPUT TO PRODUCE THE ACTUAL BAR CODE. YOU SHOULD NEVER PERFORM THIS STEP UNTIL YOUR PROGRAM HAS SUCCESSFULLY COMPILED WITHOUT ERRORS. THIS STEP IS DONE BY THE BATCH PROCESSOR AND IT MAY TAKE SEVERAL HOURS TO GET YOUR FINISHED BAR CODE.

EXECUTION OF THE THREE STEPS NECESSARY TO PRODUCE BAR CODE IS UNDER YOUR CONTROL BY SELECTION OF THE APPROPRIATE STEP FROM A MENU OF COMMANDS WHICH WILL APPEAR AT YOUR TERMINAL SHORTLY.

THE FIRST STEP IN USING THE CROSS-COMPILER IS TO PREPARE THE SOURCE CODE (YOUR PROGRAM) ON CMS DISK. THE FIRST LINE OF A SOURCE CODE FILE MUST CONTAIN THE TITLE OF THE PROGRAM THAT IS TO BE USED AS A LABEL ON THE TOP OF THE BAR CODE. THIS TITLE SHOULD HAVE NO MORE THAN 40 LETTERS. TO HELP YOU REMEMBER THAT THE LABEL OF THE PROGRAM MUST BE THE FIRST LINE, YOU MAY RECEIVE A PROMPT ASKING YOU TO ENTER THE TITLE, WHEN YOU FIRST DECLARE A NEW HP41 PROGRAM. AFTER YOU ENTER THE TITLE, YOUR TERMINAL WILL IMMEDIATELY SHIFT TO THE CMS EDITOR AND YOU WILL SEE THE TITLE AS THE FIRST LINE OF THE NEW FILE. THIS IS YOUR CUE TO ENTER THE HP41C PROGRAM THAT YOU HAVE WRITTEN. WHEN YOU EXECUTE A "FILE" COMMAND IN THE EDITOR MODE, THE TERMINAL WILL DISPLAY THE COMMAND MENU. YOU MAY THEN SELECT TO CROSS-COMPILE THE NEW PROGRAM OR ANY OTHER OPTION.

HP41C PROGRAMS AND RETURN TO CMS. IF YOU ARE EXECUTING A FUNCTION THAT WAS INVOKED FROM THE COMMAND MENU, IN MOST CASES PF13 WILL RETURN YOU TO THE MENU, AND BY PRESSING PF13 AGAIN YOU WILL RETURN TO CMS.

THIS COMMAND IS USED TO DISPLAY THE DETAILED EXPLANATION OF THE MENU COMMAND PROCESSOR AND ITS AVAILABLE COMMANDS. IF YOU HAVE QUESTIONS ABOUT THE PROCESS OF WRITING ACTUAL HP41C PROGRAMS YOU SHOULD CONSULT THE HP41 OWNER'S HANDBOOK.

THIS COMMAND IS USED TO ENTER A PROGRAM USING THE CROSS-COMPILER IN AN INTERACTIVE MODE. THE ADVANTAGE OF THIS MODE IS THAT ANY SYNTACTICAL ERRORS IN THE HP41C PROGRAM ARE IMMEDIATELY IDENTIFIED BY THE CROSS-COMPILER AND AN ERROR MESSAGE IS SHOWN ON THE SCREEN. THE DISADVANTAGE IS THAT THE USER IS TOTALLY RESPONSIBLE FOR UPPER AND LOWER CASE BEING ENTERED PROPERLY.

THIS COMMAND IS USED ONCE THE HP41C PROGRAM IS WRITTEN AND COMPILED WITHOUT ERRORS. IT SUBMITS A JOB TO MVS BATCH FOR THE PHYSICAL PRODUCTION OF THE BAR CODE.

THIS COMMAND IS USED TO DIRECT THE ATTENTION OF THE COMMAND PROCESSOR TO A NEW HP41 PROGRAM SOURCE FILE. WHEN USED TO INITIATE NEW HP41C PROGRAMS, IT AUTOMATICALLY INSURES THAT A NEW FILE IS CREATED WITH FILETYPE "HP41" AND PROMPTS THE USER FOR THE PROGRAM TITLE WHICH IS THE MANDATORY FIRST LINE OF EVERY HP41C SOURCE CODE FILE.

THIS COMMAND DISPLAYS THE FULL COMMAND MENU. IT HAS PRIMARY USE WHEN YOU FINISH AN OPERATION THAT FILLS THE SCREEN WITH TEXTUAL MATTER AND YOU RECEIVE ONLY THE PROMPT "INPUT COMMAND".

THIS COMMAND DISPLAYS "FLIST" FOR THOSE HP41C PROGRAMS THAT ARE ACTIVE ON YOUR A DISK. FROM THIS LIST, YOU CAN ERASE OLD PROGRAMS TO RELEASE DISK STORAGE, CHANGE THE NAME OF PROGRAMS, OR EXAMINE THE CONTENTS OF ANY PROGRAM.

THIS COMMAND IS USED TO PRODUCE AN "OFFLINE" COMPILE. THE PROGRAM LISTING IS AUTOMATICALLY PRINTED IN HARD COPY ON THE HIGH SPEED PRINTER. IF THE COMPILE WAS WITHOUT ERROR THE BAR CODE IS AUTOMATICALLY PRODUCED.

PF14 HELP H

PF15 ENTER E

PF16 BAR B

PF17 NEW N

PF18 DIREC D

PF19 LIST L

PF20 CCOMP C O

C	DONE=.FALSE.	HP C 00490
C		HP C 00500
C		HP C 00510
9101	READ(2,101) IPRT,PAGE	HP C 00520
101	READ(2,9101)(TITLE\$(JJ),JJ=1,25)	HP C 00530
	FORMAT(25A1)	HP C 00540
	FORMAT(12I5)	HP C 00550
	DO 5 I=1,NINST	HP C 00560
102	READ(2,102)(INST\$(J,I),J=1,6),CCDE(I),LINST(I)	HP C 00570
5	FORMAT(6A1,4X,15,10X,15)	HP C 00580
C	CONTINUE	HP C 00590
C		HP C 00600
C		HP C 00610
C		HP C 00620
C		HP C 00630
C		HP C 00640
103	READ(5,103) (TITLE\$(I),I=26,IDIM)	HP C 00650
C	FORMAT(75A1)	HP C 00660
C		HP C 00670
C		HP C 00680
C		HP C 00690
C		HP C 00700
C		HP C 00710
C	LENGTH=0	HP C 00720
C	DO 16 I=1,MAX	HP C 00730
C	(I=THE INSTR NUMBER, J=CHARACTER IN INSTR)	HP C 00740
C		HP C 00750
C		HP C 00760
C		HP C 00770
C		HP C 00780
C	ATTEMPT TO READ A TEXT STRING.	HP C 00790
C		HP C 00800
292	IF(IPRT.GE.20)WRITE(6,292) UNDER	HP C 00810
C	FORMAT('1','NEXT INSTRUCTION:',15A4,//)	HP C 00820
C		HP C 00830
C	IF(IN\$(T\$,LT,5)) 14,12,12	HP C 00840
C		HP C 00850
C	GO TO THE FOLLOWING INSTRUCTIONS IF A CHARACTER STRING FOUND	HP C 00860
C		HP C 00870
C		HP C 00880
C		HP C 00890
C		HP C 00900
C	CHECK FOR A COMMENT CARD AND/OR PAGE EJECT	HP C 00910
C		HP C 00920
C		HP C 00930
C	*** TWO "COMMENT" CHARACTERS IN POSITION 1 AND 2 OF AN INPUT	HP C 00940
C	LINE ARE CONSIDERED A MANDATORY PAGE EJECT PRAGMA. ***	HP C 00950
C		HP C 00960

C 12
268
263
C
C
C
C
13
C
269
C
C
C
9914
9916
9915
C
C
C
C
15
16
C
C
C
C
14
259

```

IF(T$(1),NE.COMENT)GOTO 13
IF(MOD(LINCNT,PAGE).EQ.0.OR.(T$(2).EQ.COMENT.AND.LT.GE.2))
1 CALL NEWPG$(LINCNT,NUMPGE,TITLE$,LTITLE,1)
LINCNT=LINCNT+1
WRITE(6,268) (T$(J),J=1,LT)
FORMAT(,110A1)
IF(IPRT.GE.10)WRITE(6,263)
FORMAT(: FOUND COMMENT CARD. NOTHING MORE DONE.)
GOTO 16

IF NOT A COMMENT, INCREMENT THE INSTRUCTION COUNTER AND PRINT
THE INSTRUCTION.
LENGTH=LENGTH+1
IF(MOD(LINCNT,PAGE).EQ.0)
1 CALL NEWPG$(LINCNT,NUMPGE,TITLE$,LTITLE,1)
LINCNT=LINCNT+1
WRITE(6,269)LENGTH,(T$(J),J=1,LT)
FORMAT(,14,.,110A1)

TRIM OFF TRAILING COMMENTS
IF(FIND$(COMMENT,1,T$,LT,LOC)) 6000,9915,9914
LT=LOC-1
IF(TRIM$(T$,LT)) 6000,9916,9915
CONTINUE
GOTO 16
CONTINUE

COMPILE THE TEXT INSTRUCTION.
IF(COMP$(T$,LT,M,ML)) 15,16,20
ERROR=.TRUE.
CONTINUE

GOTO THE FOLLOWING INSTRUCTIONS IF END OF FILE ENCOUNTERED
WRITE(6,259)
FORMAT(*****END OF FILE*****))

```

HPC 00970
HPC 00980
HPC 00990
HPC 01000
HPC 01010
HPC 01020
HPC 01030
HPC 01040
HPC 01050
HPC 01060
HPC 01070
HPC 01080
HPC 01090
HPC 01100
HPC 01110
HPC 01120
HPC 01130
HPC 01140
HPC 01150
HPC 01160
HPC 01170
HPC 01180
HPC 01190
HPC 01200
HPC 01210
HPC 01220
HPC 01230
HPC 01240
HPC 01250
HPC 01260
HPC 01270
HPC 01280
HPC 01290
HPC 01300
HPC 01310
HPC 01320
HPC 01330
HPC 01340
HPC 01350
HPC 01360
HPC 01370
HPC 01380
HPC 01390
HPC 01400
HPC 01410
HPC 01420
HPC 01430
HPC 01440

HPC01450
 HPC01460
 HPC01470
 HPC01480
 HPC01490
 HPC01500
 HPC01510
 HPC01520
 HPC01530
 HPC01540
 HPC01550
 HPC01560
 HPC01570
 HPC01580
 HPC01590
 HPC01600
 HPC01610
 HPC01620
 HPC01630
 HPC01640
 HPC01650
 HPC01660
 HPC01670
 HPC01680
 HPC01690
 HPC01700
 HPC01710
 HPC01720
 HPC01730
 HPC01740
 HPC01750
 HPC01760
 HPC01770
 HPC01780

GCTO THE FOLLOWING INSTRUCTIONS IF END COMPILATION

IF(ERROR) STOP

CALL THE BAR CODE GENERATOR.

MSAVE=M1

B1=1

IBAR=BSTR\$(M,M1,ITOT,TITLE\$)

WRITE(6,301) ITOT

FORMAT(' END HP41C CROSS COMPILE',I5,' BYTES IN TOTAL PROGRAM',)

STOP

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC\$

FORMAT(' *** STRING LENGTH ERROR *** ',A4)

MAIN\$=0

RETURN

WRITE(6,6003) FUNC\$

FORMAT(' *** FUNCTION CALL ERROR *** ',A4)

MAIN\$=0

RETURN

END

C
 C
 C
 20
 C
 C
 C
 30

301
 C
 C
 C
 C

C
 C
 C
 C
 C

6000
 6001

6002
 6003

210	B(2)=1			AIN00490
	OPERND=D2			AIN00500
				AIN00510
				AIN00520
				AIN00530
				AIN00540
230	D3=OPERND-32			AIN00550
	IF(D3) 300,310,310			AIN00560
300	B(3)=0			AIN00570
	GOTO 340			AIN00580
310	B(3)=1			AIN00590
	OPERND=D3			AIN00600
				AIN00610
				AIN00620
				AIN00630
				AIN00640
340	D4=OPERND-16			AIN00650
	IF(D4) 400,410,410			AIN00660
400	B(4)=0			AIN00670
	GOTO 450			AIN00680
410	B(4)=1			AIN00690
	OPERND=D4			AIN00700
				AIN00710
				AIN00720
				AIN00730
				AIN00740
450	D5=OPERND-8			AIN00750
	IF(D5) 500,510,510			AIN00760
500	B(5)=0			AIN00770
	GOTO 560			AIN00780
510	B(5)=1			AIN00790
	OPERND=D5			AIN00800
				AIN00810
				AIN00820
				AIN00830
				AIN00840
560	D6=OPERND-4			AIN00850
	IF(D6) 600,610,610			AIN00860
600	B(6)=0			AIN00870
	GOTO 670			AIN00880
610	B(6)=1			AIN00890
	OPERND=D6			AIN00900
				AIN00910
				AIN00920
				AIN00930
				AIN00940
				AIN00950
670	D7=OPERND-2			AIN00960
	IF(D7) 700,710,710			

700

B(7)=0
GOTO 780

710

B(7)=1
OPERND=D7

C

C

CONVERT THE EIGHTH BINARY DIGIT

C

780

D8=OPERND-1
IF(D8) 800,810,6000

800

B(8)=0
GOTO 1000

810

B(8)=1

C

C

C

WRITE OUT CONVERSION IF NECESSARY AND RETURN

C

1000

IF(IPRT.GE.20) WRITE(6,66) INOPER,(B(I),I=1,8)
FORMAT(:TRACE AIN\$ OPERAND=,I5,' BINARY= ',8I1)

66

AIN\$=0
RETURN

C

C

C

ERROR HANDLING SECTION FOLLOWS

C

C

6000

WRITE(6,6001) FUNC\$

6001

FORMAT(: **** CONVERSION ERROR ***** ',A4)

6002

WRITE(6,6002) INOPER AIN\$ OPERAND=,I5)

AIN\$=-1

RETURN

END

AIN00970
 AIN00980
 AIN00990
 AIN01000
 AIN01010
 AIN01020
 AIN01030
 AIN01040
 AIN01050
 AIN01060
 AIN01070
 AIN01080
 AIN01090
 AIN01100
 AIN01110
 AIN01120
 AIN01130
 AIN01140
 AIN01150
 AIN01160
 AIN01170
 AIN01180
 AIN01190
 AIN01200
 AIN01210
 AIN01220
 AIN01230
 AIN01240
 AIN01250
 AIN01260
 AIN01270
 AIN01280
 AIN01290
 AIN01300
 AIN01310


```

C** INTEGER FUNCTION ALPH$(A$,LA,M,M1)
C**
C** THIS FUNCTION INTERPRETS ALPHABETIC CHARACTERS INTO HP41C KEY
C** CODES.
C**
C** THE RETURN VALUE OF THE FUNCTION ALPH$ IS SET AS FOLLOWS:
C** 0 = CONTINUE TO COMPILE
C** -1 = AN ERROR IN COMPILING THE INSTRUCTION.
C**
C** *****
C** IMPLICIT INTEGER(A-Z)
C** COMMON/TEXT/IDIM,IPRT
C** COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C** LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C** LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA
C** INTEGER*2 A$(IDIM)
C** INTEGER*2 BLNK//,QUOTE/'"/
C** INTEGER*4 FUNC$/ALPH//
C** INTEGER*4 M(1)
C** INTEGER*4 FUNC$/COMP//
C** INTEGER*2 C$(60)//,#,,$,%,&,'(,*,+,.,:;,/,0,
C** 1,2,3,4,5,6,7,8,9,:;,/,0,
C** =,>?,@,A,B,C,D,E,F,G,H,
C** I,J,K,L,M,N,O,P,Q,R,S,T,
C** U,V,W,X,Y,Z[,],/,
C** 32,39,36,37,126,42,43,44,45,46,47,48,49,50,51,52,
C** 53,54,55,56,57,58,59,60,61,62,63,13,65,66,67,68,
C** 69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,
C** 85,86,87,88,89,90,94,97,98,99,100,101/
C**
C** INTEGER*4 C2(60)/
C** INTEGER*4 LC/60/
C** IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
C** FCRMAT(,TRACE,13,ALPH$,110A1}
C** IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C**
C** DO 35 I=1,LA
C** IF(A$(I).NE.QUOTE) GOTO 15
C** HAVE FOUND A QUOTE MARK--DISREGARD IF 1ST INSTR,ELSE QUIT
C** IF(I.EQ.1) GOTO 35
C** GOTO 40
C** IZ=FINDD$(A$(I),LA,C$,LC,LOC)
C** IF(IZ.NE.0)GOTO 20
C** WRITE(6,207)M1
C** FORMAT(,***** INVALID CHARACTER ***** ,5X,I5)
C** A$(I)=BLNK
C**
C** *****
C** ALP00010
C** ALP00020
C** ALP00030
C** ALP00040
C** ALP00050
C** ALP00060
C** ALP00070
C** ALP00080
C** ALP00090
C** ALP00100
C** ALP00110
C** ALP00120
C** ALP00130
C** ALP00140
C** ALP00150
C** ALP00160
C** ALP00170
C** ALP00180
C** ALP00190
C** ALP00200
C** ALP00210
C** ALP00220
C** ALP00230
C** ALP00240
C** ALP00250
C** ALP00260
C** ALP00270
C** ALP00280
C** ALP00290
C** ALP00300
C** ALP00310
C** ALP00320
C** ALP00330
C** ALP00340
C** ALP00350
C** ALP00360
C** ALP00370
C** ALP00380
C** ALP00390
C** ALP00400
C** ALP00410
C** ALP00420
C** ALP00430
C** ALP00440
C** ALP00450
C** ALP00460
C** ALP00470
C** ALP00480

```



```

20      IZ=1
      M(M1)=C2( IZ)
      IF(IPRT.GE.10)WRITE(6,208)M1,C$(IZ),C2(IZ),M(M1)
208    FORMAT(' ALPHA$',I5,' ALPHA CHARACTER',3X,A1,3X,I3,
2      M1=M1+1
      CONTINUE
      ALPH$=0
      RETURN
      ERROR HANDLING SECTION FOLLOWS
6000    WRITE(6,6001) FUNC$
6001    FORMAT(' **$ STRING LENGTH ERROR **$ ',A4)
6010    WRITE(6,6010) LA, LB, IDIM
      LB=0, I10,0      IDIM=0, I10)
      FORMAT(' LA= ',I10,' IDIM= ',I10)
      ALPH$=-1
      RETURN
      END

```


15	IBYTE=LA+1	ALP00490
	M(M1)=IBYTE	ALP00500
215	IF(IPRT:GE.20)WRITE(6,215)M1,IBYTE	ALP00510
	FORMAT(' ALP1\$,15,' LENGTH OF THIS INSTR IS',I3)	ALP00520
	M1=M1+1	ALP00530
		ALP00540
		ALP00550
		ALP00560
		ALP00570
		ALP00580
		ALP00590
211	M(M1)=240+LA	ALP00600
	IF(IPRT:GE.10)WRITE(6,211)M1,M(M1)	ALP00610
	FORMAT(' ALP1\$,15,' TEXT LENGTH INSTR',T75,I3)	ALP00620
	M1=M1+1	ALP00630
		ALP00640
		ALP00650
		ALP00660
		ALP00670
		ALP00680
		ALP00690
		ALP00700
		ALP00710
214	IF(IPRT:GE.10)WRITE(6,214)M1,M(M1)	ALP00720
	FORMAT(' ALP1\$,15,' ALPHA APPEND CHAR',T75,I3)	ALP00730
	M1=M1+1	ALP00740
	IF(LCUT\$(A\$,LA,1)) 6015,6015,50	ALP00750
		ALP00760
		ALP00770
		ALP00780
		ALP00790
50	ALP1\$=ALPH\$(A\$,LA,M,M1)	ALP00800
	RETURN	ALP00810
		ALP00820
		ALP00830
		ALP00840
		ALP00850
6000	WRITE(6,6001) FUNC\$	ALP00860
6001	FCRMT(' *** STRING LENGTH ERROR *** ',A4)	ALP00870
	WRITE(6,6010) LA,LB,IDIM	ALP00880
6010	FORMAT(' LA=',I10,' LB=',I10,' IDIM=',I10)	ALP00890
	ALP1\$=-1	ALP00900
	RETURN	ALP00910
6015	WRITE(6,6016)	ALP00920
6016	FORMAT(' **** INVALID OPERAND IN ALPHA ENTRY INSTR **** ')	ALP00930
	ALP1\$=-1	ALP00940
	RETURN	ALP00950
	END	ALP00960


```

C***      INTEGER FUNCTION COMP$(A$,LA,M,M1)
C***      COM00010
C***      COM00020
C***      COM00030
C***      COM00040
C***      COM00050
C***      COM00060
C***      COM00070
C***      COM00080
C***      COM00090
C***      COM00100
C***      COM00110
C***      COM00120
C***      COM00130
C***      COM00140
C***      COM00150
C***      COM00160
C***      COM00170
C***      COM00180
C***      COM00190
C***      COM00200
C***      COM00210
C***      COM00220
C***      COM00230
C***      COM00240
C***      COM00250
C***      COM00260
C***      COM00270
C***      COM00280
C***      COM00290
C***      COM00300
C***      COM00310
C***      COM00320
C***      COM00330
C***      COM00340
C***      COM00350
C***      COM00360
C***      COM00370
C***      COM00380
C***      COM00390
C***      COM00400
C***      COM00410
C***      COM00420
C***      COM00430
C***      COM00440
C***      COM00450
C***      COM00460
C***      COM00470
C***      COM00480

C***      THIS IS THE MASTER INSTRUCTION INTERPRETATION ROUTINE FOR THE
C***      HP41C COMPILER.  THIS ROUTINE IS USED BY BOTH THE BAR CODE
C***      GENERATOR AND THE CALCULATOR EMULATOR.
C***
C***      INSTRUCTIONS ARE PASSED TO THIS ROUTINE ONE AT A TIME IN
C***      A TEXT STRING A$.  THE ARRAY M IS THE TOTAL ARRAY OF DECIMAL
C***      INTEGER KEY CODES (MACHINE INSTRUCTIONS), AND M1 IS THE
C***      POSITION WHERE THE NEXT DECODED MACHINE INSTRUCTION WILL BE
C***      PLACED.  THUS, THE INPUT TO THIS ROUTINE IS A TEXTUAL HP41C
C***      INSTRUCTION AND THE OUTPUT IS ONE OR MORE DECIMAL KEY CODES
C***      PLACED APPROPRIATELY INTO ARRAY M.
C***
C***      THE RETURN VALUE OF THE FUNCTION COMP$ IS SET AS FOLLOWS:
C***      1 = END STATEMENT FOUND, END COMPILATION.
C***      0 = CONTINUE TO COMPILE
C***      -1 = AN ERROR IN COMPILING THE INSTRUCTION.
C***
C***      IMPLICIT INTEGER(A-Z)
C***      COMMON/TEXT/IDIM,IPRT
C***      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C***      COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C***      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C***      INTEGER *4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C***      INTEGER *2 A$(IDIM),/
C***      INTEGER *2 BLNK/,/
C***      INTEGER *2 T1$(40),T2$(40),SAV$(40),WORK$(80)
C***      INTEGER *2 S1$(40),SS1$(40),SS2$(40),SS3$(40),SS4$(40),SS5$(40)
C***      INTEGER *4 LT1,LT2,LSAV,LWORK
C***      INTEGER *4 LSS1,LS S2,LSS3,LSS4,LSS5
C***      INTEGER *4 M(1),M1
C***      INTEGER *4 FUNC$/COMP,/
C***      INTEGER *2 END(3)/E,N,D/,/
C***      RCL(3)/R,C,L/,/
C***      QUOTE/"",/
C***      APPEND/"",/
C***      LBL(3)/L,B,L/,/
C***      GTO(3)/G,T,O/,/
C***      XEQ(3)/X,E,Q/,/
C***      XRO(3)/X,R,O/,/
C***      STO(3)/S,T,O/,/
C***      IND(3)/I,N,D/,/

```

C

31	COMP\$=XEQ\$(A\$,LA,M,M1)	COM00970
	PDIGIT=.FALSE.	COM00980
	RETURN	COM00990
35		COM01000
36	IF(EQ\$(A\$,LA,XRD,3,3)) 6000,40,36	COM01010
	COMP\$=XRD\$(A\$,LA,M,M1)	COM01020
	PDIGIT=.FALSE.	COM01030
	RETURN	COM01040
40		COM01050
41	IF(EQ\$(A\$,LA,END,3,3)) 6000,45,41	COM01060
	COMP\$=END\$(A\$,LA,M,M1)	COM01070
	PDIGIT=.FALSE.	COM01080
	RETURN	COM01090
		COM01100
		COM01110
		COM01120
		COM01130
		COM01140
45	IF(A\$(1).NE.QUOTE) GOTO 50	COM01150
46	COMP\$=ALP1\$(A\$,LA,M,M1)	COM01160
	PDIGIT=.FALSE.	COM01170
	RETURN	COM01180
		COM01190
		COM01200
		COM01210
		COM01220
		COM01230
50	IF(NUMC\$(A\$,LA,IANSW) 6000,55,51	COM01240
51	COMP\$=DIGIT\$(A\$,LA,M,M1)	COM01250
	PDIGIT=.TRUE.	COM01260
	RETURN	COM01270
		COM01280
		COM01290
		COM01300
		COM01310
		COM01320
55	P1=POS\$(A\$,LA,BLNK,1,1)	COM01330
	IF(P1) 6000,65,70	COM01340
		COM01350
		COM01360
		COM01370
		COM01380
		COM01390
		COM01400
65	COMP\$=IONE\$(A\$,LA,M,M1,1)	COM01410
	PDIGIT=.FALSE.	COM01420
	RETURN	COM01430
		COM01440

	CHECK FOR ALPHABETIC ENTRY INSTRUCTION.
	CHECK FOR NUMERIC ENTRY INSTRUCTION.
	CHECK FOR CATEGORY ONE INSTR(ONE BYTE) BY LOOKING FOR BLANK
	NC BLANK IN STRING IMPLIES HAVE FOUND ONE BYTE INSTRUCTION

6090
6091

RETURN
WRITE(6,6091)
FORMAT(1) ***** ERROR
COMP\$=-1
RETURN
END

*****)

COM01930
COM01940
COM01950
COM01960
COM01970
COM01980


```

C***      INTEGER FUNCTION CON$(A$,LA,B$,LB,C$,LC)
C***      *****
C**      STRING A$ AND STRING B$ ARE CONCATENATED AND PLACED IN C$.
C**      *****
C**      IT IS FEASIBLE TO CON$(A$,LA,B$,LB,A$,LA) OR
C**      CON$(A$,LA,B$,LB,B$,LB)
C**      IN WHICH CASE THE APPROPRIATE STRING WILL BE REPLACED.
C**      *****
C**      THE NUMBER OF CHARACTERS IN THE RESULTING STRING C$ IS RETURNED
C**      AS THE VALUE OF THE FUNCTION CON$, UNLESS THERE IS A LOSS OF
C**      CHARACTERS IN WHICH CASE THE NUMBER OF LOST CHARACTERS IS
C**      RETURNED AS A NEGATIVE NUMBER.
C**      *****
C***      IMPLICIT INTEGER(A-Z)
C***      COMMON/TEXT/IDIM,IPRT
C***      INTEGER*2 A$(IDIM),B$(IDIM),C$(IDIM)
C***      INTEGER*4 FUNC$/CON,
C***      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
C***      IF(IPRT.GE.10) WRITE(6,201)LB,(B$(I),I=1,LB)
C***      FCFORMAT(,TRACE,,13,,CON$,,110A1)
C***      FFORMAT(,AND,,13,,B$,,110A1)
C***      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C***      IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000
C***      *****
C***      LOSS=0
C***      *****
C***      DETERMINE LENGTH OF RESULT
C***      *****
C***      ILC=LA+LB
C***      IF(ILC.LE.IDIM) GOTO 20
C***      LOSS=ILC-IDIM
C***      ILB=LB-LOSS
C***      ILC=IDIM
C***      IF(IPRT.GE.05) WRITE(6,202) LOSS
C***      FORMAT(,LOSS OF',I3,', CHARACTERS DURING CONCATENATION')
C***      GOTO 25
C***      *****
C***      ILB=LB
C***      ILA=LA
C***      INDEX=ILC
C***      *****

```


C	MOVE B\$ INTO C\$	CON00490
C	IF(ILB.LE.0) GOTO 40	CON00500
	IND=ILB	CON00510
	DC 35 I=1,ILB	CON00520
207	C\$(INDEX)=B\$(IND)	CON00530
	IF(IPRT.GE.30)WRITE(6,207) IND,B\$(IND),INDEX,C\$(INDEX)	CON00540
	FORMAT(' MOVE B(',I3,')= ',A1,' IS NOW C(',I3,')= ',A1)	CON00550
	IND=IND-1	CON00560
	INDEX=INDEX-1	CON00570
35	CONTINUE	CON00580
C		CON00590
C		CON00600
C		CON00610
C		CON00620
C		CON00630
C		CON00640
40	MOVE A\$ INTO C\$	CON00650
	IF(ILA.LE.0) GOTO 60	CON00660
	IND=ILA	CON00670
	DC 45 I=1,ILA	CON00680
	C\$(INDEX)=A\$(IND)	CON00690
209	IF(IPRT.GE.30)WRITE(6,209) IND,A\$(IND),INDEX,C\$(INDEX)	CON00700
	FORMAT(' MOVE A(',I3,')= ',A1,' IS NOW C(',I3,')= ',A1)	CON00710
	IND=IND-1	CON00720
	INDEX=INDEX-1	CON00730
45	CONTINUE	CON00740
C		CON00750
C		CON00760
C		CON00770
C		CON00780
60	LC=ILC	CON00790
65	IF(IPRT.GE.20)WRITE(6,203)LC,(C\$(I),I=1,LC)	CON00800
203	FORMAT(' CONCAT: LC=',I3,' ",I10A1)	CON00810
	IF(LOSS.NE.0) GOTO 70	CON00820
	CON\$=ILC	CON00830
	GOTO 75	CON00840
70	CON\$=-LOSS	CON00850
75	RETURN	CON00860
C		CON00870
C		CON00880
C		CON00890
6000	WRITE(6,6001) FUNC\$	CON00900
6001	FORMAT(' *** STRING LENGTH ERROR *** ',A4)	CON00910
	CCN\$=-1	CON00920
	RETURN	CON00930
	END	CON00940
		CON00950


```

6010 WRITE(6,6010) LA, LB, IDIM      IDIM=' ,I10)
      FORMAT(' ,I10, '
      DIGIT$=-1
      RETURN
6999 WRITE(6,602)M1
602  FORMAT(' ***** DIGIT ENTRY INSTR ERROR **', 5X, I5)
      DIGIT$=-1
      RETURN
      END

```

```

DIG00970
DIG00980
DIG00990
DIG01000
DIG01010
DIG01020
DIG01030
DIG01040
DIG01050

```



```

C**      INTEGER FUNCTION END$(A$,LA,M,M1)
C**      *****
C**      STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN END INSTRUCTION.
C**      *****
C**      THE RETURN VALUE OF THE FUNCTION END$ IS SET AS FOLLOWS:
C**      0 = CONTINUE TO COMPILE
C**      -1 = AN ERROR IN COMPILING THE INSTRUCTION.
C**      *****
C**      *****
C**      IMPLICIT INTEGER(A-Z)
C**      COMMON/TEXT/IDIM,IPRT
C**      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C**      *****
C**      CCOMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C**      *****
C**      THIS SUBROUTINE PASSES THE NUMBER OF ELEMENTS IN M VIA COMMON /CNTR
C**      *****
C**      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C**      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C**      INTEGER*2 A$(IDIM)
C**      INTEGER*2 BLNK/,/,/
C**      INTEGER*4 FUNC$/END*/
C**      INTEGER*4 M(1)
C**      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
C**      FORMAT(1,TRACE,13,END$:110A1)
C**      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C**      *****
C**      END INSTRUCTION IS THREE BYTES LONG.  INDICATE LENGTH OF INSTR.
C**      *****
C**      M(M1)=3
C**      IF(IPRT.GE.20)WRITE(6,201)M1,I,IBYTE
C**      FORMAT(1,END$,15,LENGTH OF NEXT INSTR IS,13)
C**      M1=M1+1
C**      *****
C**      LOAD PREFIX CODE FOR END INSTRUCTION. 193 IS USED INSTEAD OF 192.
C**      WHILE 192 IS THE HP STANDARD OP-CODE FOR "END", THE 193 IS USED TO
C**      ENABLE 192 TO STAND FOR AN ALPHA LABEL INSTRUCTION. THIS USAGE
C**      IS STANDARD AMONG THE HP USERS GROUP PRACTICING SYNTHETIC PROGRAM-

```



```

C      MING.
C      M(M1)=193
210    IF(IPRT.GE.10)WRITE(6,210)M1,M(M1)
      FORMAT(1,END$,15,1)INSTR',T75,I3)
      M1=M1+1
C
C      PROVIDE TWO NULL INSTRUCTIONS TO RESERVE SPACE FOR THE LINK
C      POINTERS. ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN
C      POINTERS WHICH LINK THEM ALTOGETHER INTO A LABEL CHAIN. THIS
C      CHAIN IS USED TO IDENTIFY THE POSITION OF LABELS AND PROGRAM
C      BCUNDARIES WITHIN THE HP41CV MEMORY. THE CHAIN OF LABELS IS
C      RECOMPILED BY THE WAND SOFTWARE, SO THE BYTES CONTAINING THE
C      CHAIN ARE SET TO ZERO BY THIS COMPILER.
C
C      M(M1)=0
211    IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
      FORMAT(1,END$,15,1)TRAILING NULL INSTR',T75,I3)
      M1=M1+1
C      M(M1)=0
212    IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
      FORMAT(1,END$,15,1)POINTER WILL BE RECOMPILED',T75,I3)
C
C      NOTE NUMBER OF ELEMENTS IN THE MACHINE CODE ARRAY AND SET END FLAG
C
C      S2=M1
C      M1=M1+1
C      DONE=.TRUE.
202    WRITE(6,202)S2
      FORMAT(1,1COMPI LATION ENDED:',15,1 MACHINE CODES GENERATED')
      END$=1
      RETURN
C
C      ERROR HANDLING SECTION FOLLOWS
C
C      WRITE(6,6001) FUNC$
6000    FORMAT(1,11* STRING LENGTH ERROR *** ',A4)
6001    WRITE(6,6010) LA, LB, IDIM
6010    FORMAT(1,110,1 LB=',110,1 IDIM=',110)
      END$=-1
      RETURN
      END
END000490)
END000500)
END000510)
END000520)
END000530)
END000540)
END000550)
END000560)
END000570)
END000580)
END000590)
END000600)
END000610)
END000620)
END000630)
END000640)
END000650)
END000660)
END000670)
END000680)
END000690)
END000700)
END000710)
END000720)
END000730)
END000740)
END000750)
END000760)
END000770)
END000780)
END000790)
END000800)
END000810)
END000820)
END000830)
END000840)
END000850)
END000860)
END000870)
END000880)
END000890)
END000900)
END000910)
END000920)
END000930)
END000940)
END000950)

```



```

C      IF((LENGTH.LE.LA).AND.(LENGTH.LE.LB))GOTO 10
C      IF(LA.NE.LB) GOTO 5
C      LENGTH=LA
C      GOTO 10
C
C      STRINGS CAN NOT BE EQUAL BECAUSE HAVE BEEN ASKED TO EXAMINE
C      MORE CHARACTERS THAN SMALLEST STRING IN A COMPARISON OF UNEQUAL
C      STRINGS.
C      EQ$=0
C      IF(IPRT.GE.20)WRITE(6,202)
C      RETURN
C
C      EXAMINE CHARACTERS ONE-BY-ONE TO TEST FOR EQUALITY.
C
C      DO 15 I=1,LENGTH
C      IF(IPRT.GE.30) WRITE(6,201) I,A$(I),I,B$(I)
C      FORMAT(' COMPARE A$(',I3,')=',A1,' WITH B$(',I3,')=',A1)
C      IF(A$(I).EQ.B$(I)) GOTO 15
C      EQ$=0
C      IF(IPRT.GE.20)WRITE(6,202)I
C      FORMAT(' STRINGS FOUND UNEQUAL',I3,' POSITION')
C      RETURN
C
C      CONTINUE
C
C      IF YOU GET BELOW HERE THE STRINGS WERE FOUND TO BE EQUAL
C
C      EQ$=1
C      IF(IPRT.GE.20)WRITE(6,203)
C      FORMAT(' STRINGS FOUND EQUAL')
C      RETURN
C
C      ERROR HANDLING SECTION FOLLOWS
C
C      WRITE(6,6001) FUNC$
C      FCRMAT(' *** STRING LENGTH ERROR *** ',A4)
C      WRITE(6,6010) LA,LB,IDIM
C      FCRMAT(' LA=',I10,' LB=',I10,' IDIM=',I10)
C      EQ$=-9999
C      RETURN
C      END

```

```

EQ$00490
EQ$00500
EQ$00510
EQ$00520
EQ$00530
EQ$00540
EQ$00550
EQ$00560
EQ$00570
EQ$00580
EQ$00590
EQ$00600
EQ$00610
EQ$00620
EQ$00630
EQ$00640
EQ$00650
EQ$00660
EQ$00670
EQ$00680
EQ$00690
EQ$00700
EQ$00710
EQ$00720
EQ$00730
EQ$00740
EQ$00750
EQ$00760
EQ$00770
EQ$00780
EQ$00790
EQ$00800
EQ$00810
EQ$00820
EQ$00830
EQ$00840
EQ$00850
EQ$00860
EQ$00870
EQ$00880
EQ$00890
EQ$00900
EQ$00910
EQ$00920
EQ$00930

```



```

C**      INTEGER FUNCTION FIND$(A$,LA,B$,LB,LOC)
C**      *****
C**      "FIND A$ IN B$."
C**      STRING B$ IS SEARCHED FOR THE FIRST OCCURANCE OF A MATCH WITH
C**      CHARACTER A$. A$ IS NOT ALLOWED TO BE MORE THAN ONE CHARACTER
C**      IN LENGTH.
C**      SINCE B$ IS MOST LIKELY A TABLE OF CHARACTERS, IT IS ALLOWED,
C**      AND MOST OFTEN IS OF A GREATER DIMENSION THAN IDIM, THE STANDARD
C**      STRING DIMENSION.
C**      THE VALUE OF THE FUNCTION FIND$ IS SET TO
C**      LOC (LOCATION OF FIRST MATCH IN B$) IF MATCH FOUND
C**      0 NO MATCH IS FOUND
C**      -1 IF AN ERROR IS ENCOUNTERED.
C**      *****
C**      IMPLICIT INTEGER(A-Z)
C**      COMMON/TEXT/IDIM,IPRT
C**      INTEGER*2 A$(1),B$(IDIM)
C**      INTEGER*2 OBJECT
C**      INTEGER*4 FUNC$/'FIND'/'
C**      INTEGER*4 LOC,FIND$
C**      IF(IPRT.GE.10) WRITE(6,200)LA,A$(1)
C**      FORMAT('TRACE ',I3,' FIND$ ',110A1)
C**      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C**      IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000
C**      *****
C**      OBJECT=A$(1)
C**      INDEX=1
C**      DO 25 I=1,LB
C**          IF(IPRT.GE.40) WRITE(6,211) OBJECT,I,B$(I)
C**          FORMAT(' COMPARE OBJECT=',A1,' WITH B$(',I3,')=',A1)
C**          IF(OBJECT.EQ.B$(INDEX))GOTO 30
C**          INDEX=INDEX+1
C**      CONTINUE
C**      NO MATCH FOUND
C**      LOC=0
C**      FIND$=0
C**      IF(IPRT.GE.20)WRITE(6,201)LOC

```


GOTO 75

OPERAND MUST BE A NUMERIC LOCAL LABEL

IF(I VAL\$(A\$,LA,INDEX))6015,55,55

HAVE FOUND VALID NUMERIC LOCAL LABEL, CHECK FOR TWO BYTE INSTR

IF(INDEX.GT.14.OR.INDIR) GOTO 75

PROCESS A TWO BYTE INSTRUCTION, FIRST LOAD THE LENGTH OF INSTR

IBYTE=2

M(M1)=IBYTE

IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE

M1=M1+1

HAVE FOUND VALID NUMERIC LOCAL LABEL <15, LOAD "GTO" INSTRUCTION

M(M1)=177+INDEX

IF(IPRT.GE.10)WRITE(6,213)M1,M(M1)

FORMAT(' GTO\$',I5,' TWO BYTE GTO INSTR',I75,I3)

M1=M1+1

LOAD NULL INSTR FOR TWO BYTE GTO INSTR AND RETURN

M(M1)=0

IF(IPRT.GE.10)WRITE(6,221)M1,M(M1)

M1=M1+1

GTO\$=0

RETURN

PROCESS THE GOTO INSTRUCTION (OPERAND>14)

GT000970
GT000980
GT000990
GT001000
GT001010
GT001020
GT001030
GT001040
GT001050
GT001060
GT001070
GT001080
GT001090
GT001100
GT001110
GT001120
GT001130
GT001140
GT001150
GT001160
GT001170
GT001180
GT001190
GT001200
GT001210
GT001220
GT001230
GT001240
GT001250
GT001260
GT001270
GT001280
GT001290
GT001300
GT001310
GT001320
GT001330
GT001340
GT001350
GT001360
GT001370
GT001380
GT001390
GT001400
GT001410
GT001420
GT001430
GT001440

75	M(M1)=IBYTE					GT001450
210	IF(IPRT:GE.20)WRITE(6,210)M1,IBYTE					GT001460
	FORMAT(:GT0\$,15,' LENGTH OF NEXT INSTR IS',I3)					GT001470
	M1=M1+1					GT001480
						GT001490
						GT001500
						GT001510
						GT001520
						GT001530
						GT001540
211	M(M1)=PREFIX					GT001550
	IF(IPRT:GE.10)WRITE(6,211)M1,M(M1)					GT001560
	FORMAT(:GT0\$,15,' GTO PREFIX					GT001570
	M1=M1+1					GT001580
						GT001590
						GT001600
						GT001610
						GT001620
						GT001630
						GT001640
221	IF(INDIR) GOTO 95					GT001650
	M(M1)=0					GT001660
	IF(IPRT:GE.10)WRITE(6,221)M1,M(M1)					GT001670
	FORMAT(:GT0\$,15,' NULL FOR GTO INSTR',T75,I3)					GT001680
	M1=M1+1					GT001690
						GT001700
						GT001710
						GT001720
C95	LOAD THE 2D OPERAND OF THE					GT001730
						GT001740
95	IF(INDIR) INDEX=INDEX+128					GT001750
	NOTE THAT FOR GTO IND THE HIGH ORDER BIT IS NOT SET					GT001760
	M(M1)=INDEX					GT001770
212	IF(IPRT:GE.10)WRITE(6,212)M1,M(M1)					GT001780
	FORMAT(:GT0\$,15,' GTO 2D OPERAND					GT001790
	M1=M1+1					GT001800
	GTO\$=0					GT001810
	RETURN					GT001820
						GT001830
						GT001840
						GT001850
						GT001860
						GT001870
						GT001880
80	PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE					GT001890
						GT001900
	K=0					GT001910
	P2=POS\$(A\$,LA,QUOTE,1,2)					GT001920
	IF(P2) 6015,120,85					

C		CHECK AFTER LAST QUOTE FOR BOGUS CHARACTERS	GT001930
C			GT001940
C			GT001950
85		LEFT=LA-P2	GT001960
C		IF(LEFT) 6015,100,6015	GT001970
C			GT001980
C		DELETE THE ENDING QUOTE BY TRUNCATING THE STRING ONE CHAR	GT001990
C			GT002000
C			GT002010
C			GT002020
100		LA=LA-1	GT002030
C			GT002040
C			GT002050
C			GT002060
C		SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL	GT002070
C		(LINE 120 ACCOUNTS FOR BEGINNING QUOTE STILL ON STRING)	GT002080
C			GT002090
120		LENGTH=LA-1	GT002100
C		FOR GTO H=2	GT002110
		H=2	GT002120
		IBYTE=H+LENGTH	GT002130
		M(M1)=IBYTE	GT002140
		IF(IIPRT.GE.20)WRITE(6,210)M1,IBYTE	GT002150
		M1=M1+1	GT002160
			GT002170
C			GT002180
C			GT002190
C		HAVE FOUND VALID ALPHANUMERIC LABEL, LCAD "GTO" INSTRUCTION	GT002200
C			GT002210
C		PREFIX=29	GT002220
C		FOR GTO PREFIX=29	GT002230
		FOR LBL PREFIX=192	GT002240
		FOR XEQ PREFIX=30	GT002250
214		M(M1)=PREFIX	GT002260
		IF(IIPRT.GE.10)WRITE(6,214)M1,M(M1)	GT002270
		FORMAT(: GTO\$,15,' ALPHA GTO INSTR',I75,I3)	GT002280
		M1=M1+1	GT002290
C			GT002300
C			GT002310
C		SET INDICATOR FOR NUMBER OF ALPHA CHARS IN LABEL	GT002320
C			GT002330
C			GT002340
C		U=240	GT002350
C		FOR GTO U=240	GT002360
		FOR LBL U=241	GT002370
		FOR XEQ U=240	GT002380
		M(M1)=U+LENGTH	GT002390
		IF(IIPRT.GE.10)WRITE(6,216)M1,M(M1)	GT002400
		FORMAT(: GTO\$,15,' LENGTH CODE ALPH GTO',I75,I3)	
216		M1=M1+1	

60	IM=IM+1	IN\$00490
65	CONTINUE	IN\$00500
	IF(IM.EQ.0) GOTO 70	IN\$00510
207	IF(IPRT.GE.20) WRITE(6,207) IM	IN\$00520
	FORMAT(' FOUND',I3,' TRAILING BLANKS IN INPUT STRING')	IN\$00530
	IF(IM.NE.80) GOTO 70	IN\$00540
	LA=0	IN\$00550
	IN\$=0	IN\$00560
	IF(IPRT.GE.20) WRITE(6,208)	IN\$00570
208	FORMAT(' FOUND INPUT STRING IS ALL BLANKS')	IN\$00580
	IF(IPRT.GE.20) WRITE(6,200)LA,(A\$(I),I=1,LA)	IN\$00590
	RETURN	IN\$00600
70	IEND=80-IM	IN\$00610
		IN\$00620
		IN\$00630
		IN\$00640
		IN\$00650
		IN\$00660
		IN\$00670
		IN\$00680
		IN\$00690
		IN\$00700
		IN\$00710
10	CONTINUE	IN\$00720
15	IF(IM.EQ.0) GOTO 25	IN\$00730
	IF(IPRT.GE.20) WRITE(6,211) IM	IN\$00740
211	FORMAT(' FOUND',I3,' LEADING BLANKS IN INPUT STRING')	IN\$00750
25	IBEG=1+IM	IN\$00760
		IN\$00770
		IN\$00780
		IN\$00790
		IN\$00800
		IN\$00810
		IN\$00820
		IN\$00830
		IN\$00840
		IN\$00850
216	IF(IPRT.GE.10) WRITE(6,216) LOSS	IN\$00860
	FORMAT(' STRING TOO LONG FOR MAX STRING LENGTH. LOST',I3)	IN\$00870
	LA=IDIM	IN\$00880
	IEND=IEND-LOSS	IN\$00890
		IN\$00900
		IN\$00910
		IN\$00920
		IN\$00930
		IN\$00940
		IN\$00950
		IN\$00960

	CHECK CARD FOR LEADING BLANKS	
	IM=0	
	DO 10 I=1,IEND	
	IF(CARD(I).NE.BLNK) GOTO 15	
	IM=IM+1	
	CONTINUE	
10	IF(IM.EQ.0) GOTO 25	
15	IF(IPRT.GE.20) WRITE(6,211) IM	
211	FORMAT(' FOUND',I3,' LEADING BLANKS IN INPUT STRING')	
25	IBEG=1+IM	
	DETERMINE LENGTH OF INPUT STRING	
	LA=IEND-IBEG+1	
	IF(LA.LE.IDIM) GOTO 30	
	LOSS=LA-IDIM	
	IF(IPRT.GE.10) WRITE(6,216) LOSS	
	FORMAT(' STRING TOO LONG FOR MAX STRING LENGTH. LOST',I3)	
	LA=IDIM	
	IEND=IEND-LOSS	
	TRANSFER THE CARD CHARACTERS TO THE INPUT STRING.	
	INDEX=1	
DC 85	I=IBEG,IEND	
	A\$(INDEX)=CARD(I)	
30		

209	IF(IPRT,GE,30) WRITE(6,209) I,CARD(I),INDEX,A\$(INDEX)	IN\$00970
	FORMAT(,MOVE CARD(,I3,)=,A1, IS NOW C(,I3,)=,A1)	IN\$00980
	INDEX=INDEX+1	IN\$00990
85	CONTINUE	IN\$01000
C		IN\$01010
C		IN\$01020
C		IN\$01030
C		IN\$01040
	CHECK FOR STRING ERROR AND RETURN	IN\$01050
200	IF(IPRT,GE,20) WRITE(6,200)LA,(A\$(I),I=1,LA)	IN\$01060
	FORMAT(,TRACE ,I3, IN\$: ,110A1)	IN\$01070
	IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000	IN\$01080
	IN\$=LA	IN\$01090
	RETURN	IN\$01100
C		IN\$01110
C		IN\$01120
C		IN\$01130
C		IN\$01140
999	HANDLE END OF FILE CONDITION	IN\$01150
	ECFILE(IN)=.TRUE.	IN\$01160
	IN\$=-1	IN\$01170
	LA=0	IN\$01180
215	IF(IPRT,GE,20) WRITE(6,215)	IN\$01190
	FCRMT(,END OF FILE ENCOUNTERED)	IN\$01200
	RETURN	IN\$01210
C		IN\$01220
C		IN\$01230
C		IN\$01240
C		IN\$01250
	ERROR HANDLING SECTION FOLLOWS	IN\$01260
6000	WRITE(6,6001) FUNC\$	IN\$01270
6001	FORMAT(,*** STRING LENGTH ERROR *** ,A4)	IN\$01280
	IN\$=-1	IN\$01290
	RETURN	IN\$01300
	END	IN\$01310

C		ERROR HANDLING SECTION FOLLOWS	ION00970
C	6000	WRITE(6,6001) FUNC\$	ION00980
6001		FCRMT(, **, STRING LENGTH ERROR **, A4)	ION00990
		WRITE(6,6010) LA, LB, IDIM	ION01000
6010		FORMAT(, LA=, I10, , LB=, I10, , IDIM=, I10)	ION01010
		ICNE\$=-1	ION01020
		RETURN	ION01030
6020		WRITE(6,6021)	ION01040
6021		FORMAT(, **, LENGTH OF INSTRUCTION DOES NOT MATCH NUMBER OPERANDS)	ION01050
		ICNE\$=-1	ION01060
		RETURN	ION01070
		END	ION01080
			ION01090


```

IRD00490
IRD00500
IRD00510
IRD00520
IRD00530
IRD00540
IRD00550
IRD00560
IRD00570
IRD00580
IRD00590
IRD00600
IRD00610
IRD00620
IRD00630
IRD00640

```

```

C
C
C
C
6000
6001
6007
6008
2
END

VAL=IRDR$
IF(IPRT.GE.20)WRITE(6,217) VAL
RETURN

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001)
FORMAT(: ***) END OF FILE DETECTED *** ',A4)
IRDR$=-1
STOP
WRITE(6,6008)(A$(I),I=1,LA)
FORMAT(: ***) ATTEMPT TO FIND INTG VALUE OF ALPHABETIC STRING:*/
FORMAT(: ',110A1)
STOP
END

```


194

ITW00970
ITW00980
ITW00990
ITW01000
ITW01010
ITW01020
ITW01030
ITW01040
ITW01050
ITW01060
ITW01070
ITW01080
ITW01090
ITW01100
ITW01110
ITW01120
ITW01130
ITW01140
ITW01150
ITW01160
ITW01170
ITW01180
ITW01190
ITW01200
ITW01210

```

RETURN
C
C
C
C
C
6000
6001
6010
6015
6016
6020
6021
6030
6031

ERROR HANDLING SECTION FOLLOWS

WRITE(6,6001) FUNC$
FORMAT(,**) STRING LENGTH ERROR ***,A4)
WRITE(6,6010) LA, LB, IDIM
FORMAT(, LA=, I10, LB=, I10, IDIM=, I10)
ITWO$=-1
RETURN
WRITE(6,6016)
FORMAT(,**) INVALID SECOND OPERAND IN INSTR *****
ITWO$=-1
RETURN
WRITE(6,6021)
FCRMT(,**) FOUND THREE OPERANDS, EXPECTING IND *****
ITWO$=-1
RETURN
WRITE(6,6031)
FORMAT(,**) FOUND SECOND OPERAND BLANK *****
ITWO$=-1
RETURN
END

```


660	IVAL\$=SIGN*IFN	IVA00490
	VAL=IVAL\$	IVA00500
	IF(IPRT.GE.20)WRITE(6,217) VAL	IVA00510
	RETURN	IVA00520
C		IVA00530
C	ERROR HANDLING SECTION FOLLOWS	IVA00540
C		IVA00550
6000	WRITE(6,6001) FUNC\$	IVA00560
6001	FORMAT('*** STRING LENGTH ERROR *** ',A4)	IVA00570
	IVAL\$=-1	IVA00580
	STOP	IVA00590
	WRITE(6,6008)(A\$(I),I=1,LA)	IVA00600
6007	FORMAT('*** ATTEMPT TO FIND REAL VALUE OF ALPHABETIC STRING: ')	IVA00610
6008	2 FORMAT(' ',110A1)	IVA00620
	STOP	IVA00630
	END	IVA00640
		IVA00650

20	IF(LA-2) 25,50,6015	LBL00490
C		LBL00500
C		LBL00510
C		LBL00520
C		LBL00530
25	DO 30 I=1,26	LBL00540
	INDEX=1	LBL00550
	IF(A\$(1).EQ.LABEL(I)) GOTC 35	LBL00560
	CONTINUE	LBL00570
30		LBL00580
C	WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND	LBL00590
C	GOTO 6015	LBL00600
C		LBL00610
C		LBL00620
C		LBL00630
C		LBL00640
C	HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPER-	LBL00650
C	AND. THEN GOTO PROCESS A TWO BYTE INSTRUCTION.	LBL00660
35	INDEX=INDEX+101	LBL00670
C		LBL00680
C		LBL00690
C		LBL00700
C		LBL00710
C		LBL00720
C		LBL00730
40	IBYTE=2	LBL00740
	M(M1)=IBYTE	LBL00750
	IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE	LBL00760
210	FORMAT(' LBL\$',I5,' LENGTH OF NEXT INSTR IS',I3)	LBL00770
	M1=M1+1	LBL00780
C		LBL00790
C		LBL00800
C		LBL00810
C	LOAD TWO BYTE LBL INSTR (EITHER SINGLE CHAR ALPHA OR 2 DIGIT NUM)	LBL00820
		LBL00830
211	M(M1)=207	LBL00840
	IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)	LBL00850
	FORMAT(' LBL\$',I5,' TWO BYTE LBL INSTR',T75,I3)	LBL00860
	M1=M1+1	LBL00870
	M(M1)=INDEX	LBL00880
212	IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)	LBL00890
	FORMAT(' LBL\$',I5,' LBL 20 OPERAND',T75,I3)	LBL00900
	M1=M1+1	LBL00910
	LBL\$=0	LBL00920
	RETURN	LBL00930
C		LBL00940
C		LBL00950
C		LBL00960

C	LENGTH OF OPERAND IMPLIES MUST BE A NUMERIC LOCAL LABEL	LBL000970
C		LBL000980
50	IF(I VAL\$(A\$,LA,INDEX))6015,55,55	LBL000990
C		LBL001000
C		LBL001010
C	HAVE FOUND VALID NUMERIC LOCAL LABEL, CHECK FOR ONE BYTE INSTR	LBL001020
C		LBL001030
55	IF(INDEX.GT.14) GOTO 40	LBL001040
C		LBL001050
C		LBL001060
C		LBL001070
C	PROCESS A ONE BYTE INSTRUCTION, FIRST LOAD THE LENGTH OF INSTR	LBL001080
C		LBL001090
	IBYTE=1	LBL001100
	M(M1)=IBYTE	LBL001110
	IF(I PR T.GE.20)WRITE(6,210)M1,IBYTE	LBL001120
	M1=M1+1	LBL001130
C		LBL001140
C		LBL001150
C		LBL001160
C	HAVE FOUND VALID NUMERIC LOCAL LABEL <15, LOAD "LBL" INSTRUCTION	LBL001170
		LBL001180
213		LBL001190
	M(M1)=1+INDEX	LBL001200
	IF(I PR T.GE.10)WRITE(6,213)M1,M(M1)	LBL001210
	FORMAT(' LBL\$',15,' ONE BYTE LBL INSTR',T75,I3)	LBL001220
	M1=M1+1	LBL001230
	LBL\$=0	LBL001240
	RETURN	LBL001250
C		LBL001260
C		LBL001270
C	PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE	LBL001280
C		LBL001290
80		LBL001300
	K=0	LBL001310
	P2=POS\$(A\$,LA,QUOTE,1,2)	LBL001320
	IF(P2) 6015,120,85	LBL001330
C		LBL001340
C		LBL001350
C	LCOK FOR ANOTHER QUOTE	LBL001360
C		LBL001370
C		LBL001380
85		LBL001390
	PL=P2+1	LBL001400
	P4=POS\$(A\$,LA,QUOTE,1,PL)	LBL001410
	IF(P4) 6015,95,90	LBL001420
C		LBL001430
C		LBL001440


```

C          FCUND ANOTHER (THIRD OR MORE) QUOTE
C
C 90      P2=P4
C          GOTO 85
C
C          CHECK AFTER LAST QUOTE FOR KEY ASSIGNMENT CODE
C
C 95      LEFT=LA-P2
C          IF(LEFT) 6015,100,105
C
C 100     LA=LA-1
C          GOTO 120
C
C 105     PSTART=P2+1
C          CALL SEG$(A$,LA,SS1$,LSS1,PSTART,LEFT)
C          K=IVAL$(SS1$,LSS1,NO)
C          LA=LA-LEFT-1
C
C          SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL
C
C 120     LENGTH=LA-1
C          IBYTE=4+LENGTH
C          M(M1)=IBYTE
C          IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
C          M1=M1+1
C
C          HAVE FOUND VALID ALPHANUMERIC LABEL,      LOAD "LBL" INSTRUCTION
C
C          M(M1)=192
C          IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)
C          FORMAT('  LBL$',I5,' ALPHA  LBL INSTR',T75,I3)
C          M1=M1+1
C
C 214
C
C          PROVIDE ONE NULL INSTRUCTION TO RESERVE SPACE FOR THE LINK
C          PCINTER.  ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN
C          POINTERS WHICH LINK THEM ALTOGETHER INTO A LABEL CHAIN.  THIS
C          CHAIN IS USED TO IDENTIFY THE POSITION OF LABELS AND PROGRAM
C          BOUNDARIES WITHIN THE HP41CV MEMORY.  THE CHAIN OF LABELS IS
C          RECOMPILED BY THE WAND SOFTWARE, SO THE BYTES CONTAINING THE

```


C	CHAIN ARE SET TO ZERO BY THIS COMPILER.	LBL01930
C		LBL01940
215	M(M1)=0	LBL01950
C	IF(IPRT,GE.10)WRITE(6,215)M1,M(M1)	LBL01960
C	FORMAT(, LBL\$,15,, TRAILING NULL INSTR',T75,I3)	LBL01970
C	M1=M1+1	LBL01980
C		LBL01990
C		LBL02000
C		LBL02010
C		LBL02020
C		LBL02030
216	M(M1)=241+LENGTH	LBL02040
C	IF(IPRT,GE.10)WRITE(6,216)M1,M(M1)	LBL02050
C	FORMAT(, LBL\$,15,, LENGTH CODE ALPH LBL',T75,I3)	LBL02060
C	M1=M1+1	LBL02070
C		LBL02080
C		LBL02090
C		LBL02100
C		LBL02110
C		LBL02120
C		LBL02130
C		LBL02140
217	M(M1)=0	LBL02150
C	IF(IPRT,GE.10)WRITE(6,217)M1,M(M1)	LBL02160
C	FORMAT(, LBL\$,15,, NULL KEY ASSIGNMENT ',T75,I3)	LBL02170
C	M1=M1+1	LBL02180
C	GOTO 140	LBL02190
C		LBL02200
C	SINCE K=0 IMPLIES A KEY ASSIGNMENT TO BE MADE	LBL02210
130	K1=0	LBL02220
C	IF(K.GT.0) GOTO 135	LBL02230
C	K1=8	LBL02240
135	K=IABS(K)	LBL02250
C	A1=K/10	LBL02260
C	B1=MOD(K,10)	LBL02270
C	K=16*(B1-1)+A1+K1	LBL02280
C	M(M1)=K	LBL02290
218	IF(IPRT,GE.10)WRITE(6,218)M1,M(M1)	LBL02300
C	FORMAT(, LBL\$,15,, LBL KEY ASSIGNMENT ',T75,I3)	LBL02310
C	M1=M1+1	LBL02320
C		LBL02330
C		LBL02340
C		LBL02350
C		LBL02360
C		LBL02370
C		LBL02380
140	LA=LENGTH+1	LBL02390
C	LBL\$=ALPH\$(A\$,LA,M,M1)	LBL02400
C	RETURN	

C
C
C
C
6000
6001
6010
6015
6016

```

ERROR HANDLING SECTION FOLLOWS
WRITE(6,6001) FUNC$
FORMAT(1,6001) STRING LENGTH ERROR *** ',A4)
WRITE(6,6010) LA, LB, IDIM
FORMAT(1,6010) LA= ', I10, ' IDIM= ', I10)
LBL$=-1
RETURN
WRITE(6,6016)
FCRMT(1,6016) INVALID SECOND OPERAND IN LBL INSTR *****
LBL$=-1
RETURN
END

```

LBL02410
LBL02420
LBL02430
LBL02440
LBL02450
LBL02460
LBL02470
LBL02480
LBL02490
LBL02500
LBL02510
LBL02520
LBL02530
LBL02540
LBL02550


```

MEM000010)
MEM000020)
MEM000030)
MEM000040)
MEM000050)
MEM000060)
MEM000070)
MEM000080)
MEM000090)
MEM000100)
MEM000110)
MEM000120)
MEM000130)
MEM000140)
MEM000150)
MEM000160)
MEM000170)
MEM000180)
MEM000190)
MEM000200)
MEM000210)
MEM000220)
MEM000230)
MEM000240)
MEM000250)
MEM000260)
MEM000270)
MEM000280)
MEM000290)
MEM000300)
MEM000310)
MEM000320)
MEM000330)
MEM000340)
MEM000350)
MEM000360)
MEM000370)
MEM000380)
MEM000390)
MEM000400)
MEM000410)
MEM000420)
MEM000430)
MEM000440)
MEM000450)
MEM000460)
MEM000470)
MEM000480)

INTEGER FUNCTION MEM$(A$,LA,M,M1,WHICH)
*****
STRING A$ CONTAINS AN MEMORY INSTRUCTION, EITHER AN STO OR A
RCL
THE RETURN VALUE OF THE FUNCTION MEM$ IS SET AS FOLLOWS:
    0 = CONTINUE TO COMPILE
   -1 = AN ERROR IN COMPILING THE INSTRUCTION.
*****
IMPLICIT INTEGER(A-Z)
COMMON/TEXT/IDIM,IPRT
COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
INTEGER*2 A$(IDIM),/
INTEGER*2 BLNK/,/
INTEGER*2 IND(3)/I,N,D,/
INTEGER*2 C$(5)/T,Z,Y,X,L,/
INTEGER*4 LC/5/
INTEGER*4 WHICH(5)
INTEGER*4 FUNC$/MEM'/
INTEGER*4 M(1)
IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
FORMAT('TRACE ',I3,' MEM$',110A1)
IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000

STRIP STRING OF "MEM" CHARACTERS.

IF(LCUT$(A$,LA,3)) 6015,6015,7
IF(TRIM$(A$,LA)) 6015,6015,10

ESTABLISH MOST LIKELY INSTR LENGTH AND PREFIX

IBYTE=2
PREFIX=WHICH(1)
H=WHICH(2)

CHECK FOR INDIRECT ADDRESS
```


20	P6=POS\$(A\$,LA,IND,3,1)	MEM00490
	IF(P6) 6000,40,21	MEM00500
21	INDIR=.TRUE.	MEM00510
22	IF(LCUT\$(A\$,LA,3)) 6000,6015,22	MEM00520
	IF(TRIM\$(A\$,LA)) 6015,6015,40	MEM00530
		MEM00540
		MEM00550
		MEM00560
		MEM00570
		MEM00580
		MEM00590
40	IF(NUMC\$(A\$,LA,IANSW)) 6015,45,60	MEM00600
		MEM00610
		MEM00620
		MEM00630
		MEM00640
		MEM00650
		MEM00660
		MEM00670
		MEM00680
		MEM00690
207	WRITE(6,207)A\$(1)	MEM00700
	FORMAT(1) ***** INVALID CHARACTER	MEM00710
	MEM\$=-1	MEM00720
	RETURN	MEM00730
46	POSTFX=IZ+111	MEM00740
	GOTO 100	MEM00750
		MEM00760
		MEM00770
		MEM00780
		MEM00790
		MEM00800
60	POSTFX=IVAL\$(A\$,LA,VAL)	MEM00810
		MEM00820
		MEM00830
		MEM00840
		MEM00850
		MEM00860
		MEM00870
		MEM00880
		MEM00890
		MEM00900
		MEM00910
		MEM00920
		MEM00930
		MEM00940
		MEM00950
		MEM00960

20	IF(LA.GT.1) GOTO 6015
	IZ=FINDD\$(A\$(1),LA,C\$,LC,LCC)
	IF(IZ.NE.0) GOTO 46
	WRITE(6,207)A\$(1)
	FORMAT(1) ***** INVALID CHARACTER
	MEM\$=-1
	RETURN
46	POSTFX=IZ+111
	GOTO 100

20	PROCESS NUMERIC POSTFIX OPERANDS
	POSTFX=IVAL\$(A\$,LA,VAL)

60	CHECK FOR ONE BYTE VERSUS TWO BYTE NUMERIC OPERANDS
	IF((POSTFX.GT.15).CR.(INDIR)) GOTO 75

20	PROCESS ONE BYTE NUMERIC SECOND OPRANDS
	IBYTE=1
	PREFIX=POSTFX+H
	GOTO 100

C	PROCESS TWO BYTE NUMERIC OPERANDS	MEM00970
C		MEM00980
C		MEM00990
75	CONTINUE	MEM01000
C		MEM01010
C		MEM01020
C		MEM01030
100	SET THE LENGTH OF THE INSTRUCTION	MEM01040
		MEM01050
215	M(M1)=IBYTE	MEM01060
	IF(IPT:GE.20)WRITE(6,215)M1,IBYTE	MEM01070
	FORMAT(: MEM\$,15,' LENGTH OF THIS INSTR IS',I3)	MEM01080
	M1=M1+1	MEM01090
C		MEM01100
C		MEM01110
C		MEM01120
C		MEM01130
		MEM01140
211	M(M1)=PREFIX	MEM01150
	IF(IPT:GE.10)WRITE(6,211)M1,(WHICH(I),I=3,5),M(M1)	MEM01160
	FORMAT(: MEM\$,15,' ',3A1,' INSTR',T75,I3)	MEM01170
	M1=M1+1	MEM01180
C		MEM01190
C		MEM01200
C		MEM01210
C		MEM01220
		MEM01230
		MEM01240
		MEM01250
		MEM01260
221	IF(IPT:GE.10)WRITE(6,221)M1,(WHICH(I),I=3,5),M(M1)	MEM01270
	FORMAT(: MEM\$,15,' ',3A1,' INSTR POSTFIX',T75,I3)	MEM01280
	M1=M1+1	MEM01290
C		MEM01300
C		MEM01310
C		MEM01320
C		MEM01330
C		MEM01340
125	MEM\$=0	MEM01350
	RETURN	MEM01360
C		MEM01370
C		MEM01380
C		MEM01390
		MEM01400
6000	WRITE(6,6001) FUNC\$	MEM01410
6001	FORMAT(: **\$ STRING LENGTH ERROR *** ',A4)	MEM01420
	WRITE(6,6010) LA, LB, IDIM	MEM01430
6010	FORMAT(: LA=,I10,' LB=,I10,' IDIM=,I10)	MEM01440

MEM01450
MEM01460
MEM01470
MEM01480
MEM01490
MEM01500
MEM01510

MEM\$=-1
RETURN
6015 WRITE(6,6016)
6016 ***
INVALID SECOND OPERAND IN MEM INSTR ***
FCRMAT(,)
MEM\$=-1
RETURN
END

NEW00490
 NEW00500
 NEW00510
 NEW00520
 NEW00530
 NEW00540
 NEW00550
 NEW00560
 NEW00570
 NEW00580
 NEW00590
 NEW00600
 NEW00610
 NEW00620

```

C      EXIT
C      NEWPG$=NUNPGE
C      RETURN
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT('*** PAGE OUTPUT   ERROR *** ',A4)
      NEWPG$=-1
      RETURN
      ,END
  
```



```

202      IANSW=1
        NUMC$=1
        IF(IPRT,GE,20) WRITE(6,202)
        FCFORMAT(,STRING FOUND TO BE NUMERIC.)
        RETURN
C
C
C
C
6000      ERROR HANDLING SECTION FOLLOWS
6001      WRITE(6,6001) FUNC$
        FORMAT(,***,STRING LENGTH ERROR ***,A4)
6010      WRITE(6,6010) LA,LB,IDIM LB=,ILO, IDIM=,IIO)
        FCFORMAT(,LA=,ILO,,LB=,IIO,,IDIM=,IIO)
        NUMC$=-1
        RETURN
        END

```



```

C** INTEGER FUNCTION PARS$(A$,LA,B$,LB)
C**
C** STRING A$ IS SEARCHED FOR THE OCCURANCE OF THE 1ST NON-LEADING
C** BLANK. THEN A$ IS SPLIT INTO TWO SUBSTRINGS, THE LEADING
C** TOKEN (FIRST WORD) IS PLACED IN B$ AND THE REMAINDER IS PLACED
C** IN A$.
C**
C** THE NUMBER OF NON-BLANK CHARACTERS REMAINING IN A$ AFTER THE
C** REMOVAL OF THE FIRST WORD IS RETURNED AS THE FUNCTION VALUE
C** POS$.
C**
C** AN ATTEMPT TO PARSE A NULL STRING WILL RESULT IN A SPECIAL
C** CHARACTER BEING PLACED IN THE 1ST POSITION OF A$. A SUBSEQUENT
C** ATTEMPT TO REPARSE THIS STRING WILL RESULT IN A FATAL ERROR.
C** THIS FEATURE IS INTENDED TO PREVENT UNCONTROLLED LOOPING OF THE
C** PARSE FUNCTION ON A NULL STRING. THE SPECIAL CHARACTER USED
C** IS KNOWN ONLY TO THIS ROUTINE AND MAY BE USED AS A REGULAR
C** CHARACTER FOR IN ANY STRING OF LENGTH GREATER THAN ZERO.
C**
C** IMPLICIT INTEGER(A-Z)
C** COMMON/TEXT/IDIM,IPRT
C** INTEGER*2 A$(IDIM),B$(IDIM)
C** INTEGER*2 BLNK//
C** INTEGER*2 HALT//
C** INTEGER*4 FUNC$/PARS$/
C** IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
C** FORMAT(1,TRACE,13,PARS$,110A1)
C** IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C**
C** CHECK TO SEE IF INPUT STRING IS NULL
C**
C** IF(LA.NE.0) GOTO 5
C** IF(IPRT.GE.20) WRITE(6,204)
C** FORMAT(1,ATTEMPTED TO PARSE A NULL STRING.1)
C** IF(A$(1).EQ.HALT) GOTO 6005
C** A$(1)=HALT
C** PARS$=0
C** LB=0
C** RETURN
C**
C** TRIM THE INPUT STRING OF LEADING BLANKS
C**

```


5	IM=0			PAR00490
	DO 10 I=1,LA			PAR00500
	IF(A\$(I).NE.BLNK) GOTO 15			PAR00510
	IM=IM+1			PAR00520
10	CONTINUE			PAR00530
15	IF(IM.EQ.0) GOTO 25			PAR00540
	IF(IPRT.GE.20) WRITE(6,201) IM			PAR00550
201	FORMAT(' FOUND',I3,' LEADING BLANKS IN INPUT STRING')			PAR00560
	IF(IM.LT.LA) GOTO 20			PAR00570
	LA=0			PAR00580
	LB=0			PAR00590
	PARS\$=0			PAR00600
202	IF(IPRT.GE.20) WRITE(6,202)			PAR00610
	FORMAT(' FOUND INPUT STRING IS ALL BLANKS')			PAR00620
	RETURN			PAR00630
20	LEFT=LA-IM			PAR00640
	DO 23 I=1,LEFT			PAR00650
23	A\$(I)=A\$(I+IM)			PAR00660
	CONTINUE			PAR00670
	LA=LEFT			PAR00680
				PAR00690
				PAR00700
				PAR00710
				PAR00720
25	LB=0			PAR00730
	DO 30 I=1,LA			PAR00740
	IF(A\$(I).EQ.BLNK) GOTO 35			PAR00750
	LB=LB+1			PAR00760
30	CONTINUE			PAR00770
				PAR00780
				PAR00790
				PAR00800
				PAR00810
				PAR00820
35	DO 45 I=1,LB			PAR00830
	B\$(I)=A\$(I)			PAR00840
45	CONTINUE			PAR00850
	IF(IPRT.GE.20) WRITE(6,205) LB,(B\$(I),I=1,LB)			PAR00860
205	FORMAT(' PARS\$ FOUND TOKEN',I3,' ",60A1)			PAR00870
				PAR00880
				PAR00890
				PAR00900
				PAR00910
				PAR00920
				PAR00930
				PAR00940
				PAR00950
50	DO 55 I=1,LEFT			PAR00960

55	A\$(I)=A\$(I+LB+1)	PAR00970
	CONTINUE	PAR00980
	LA=LEFT	PAR00990
		PAR01000
	CHECK \$A FOR TRAILING BLANKS	PAR01010
		PAR01020
	IF(LA.EQ.0) GOTO 75	PAR01030
58	IM=0	PAR01040
	DO 60 I=1,LA	PAR01050
	INDEX=LA-I+1	PAR01060
	IF(A\$(INDEX).NE.BLNK) GOTO 65	PAR01070
	IM=IM+1	PAR01080
	CONTINUE	PAR01090
60	IF(IM.EQ.0) GOTO 75	PAR01100
65	IF(IPRT.GE.20) WRITE(6,207) IM	PAR01110
207	FORMAT(' FOUND',I3,' TRAILING BLANKS IN INPUT STRING')	PAR01120
	IF(IM.LT.LA) GOTO 70	PAR01130
	LA=0	PAR01140
	PAR\$=0	PAR01150
208	IF(IPRT.GE.20) WRITE(6,208)	PAR01160
	FORMAT(' FOUND REMAINING STRING IS ALL BLANKS')	PAR01170
	RETURN	PAR01180
70	LA=LA-IM	PAR01190
75	PAR\$=LA	PAR01200
209	IF(IPRT.GE.20.AND.LA.EQ.0) WRITE(6,209)	PAR01210
	FORMAT(' REMAINING STRING AFTER PARSE FUNCTION IS NULL')	PAR01220
	RETURN	PAR01230
		PAR01240
		PAR01250
		PAR01260
		PAR01270
	ERROR HANDLING SECTION FOLLOWS	PAR01280
		PAR01290
6000	WRITE(6,6001) FUNC\$	PAR01300
6001	FORMAT(' *** STRING LENGTH ERROR *** ',A4)	PAR01310
	PAR\$=-1	PAR01320
	RETURN	PAR01330
6005	WRITE(6,6006)	PAR01340
6006	FORMAT(' *** FATAL ERROR: ATTEMPTED TO PARSE A NULL STRING TWICE')	PAR01350
	STOP	PAR01360
	END	PAR01370


```

C***      INTEGER FUNCTION POS$(A$,LA,B$,LB,LSTART)
C***      *****
C**      THE STRING A$ IS SEARCHED FOR THE OCCURANCE OF THE SUBSTRING
C**      B$, AND THE POSITION OF THE 1ST CHARACTER OF A$ WHERE B$ IS
C**      A MATCH IS ASSIGNED TO POS$. THE SEARCH BEGINS AT LSTART
C**      IN A$. A ZERO IS RETURNED IF NO MATCH IS FOUND. A NEGATIVE
C**      NUMBER IS RETURNED UPON AN ERROR.
C***      *****
C***      IMPLICIT INTEGER(A-Z)
C***      COMMON/TEXT/IDIM,IPRT
C***      INTEGER*2 A$(IDIM),B$(IDIM)
C***      INTEGER*4 FUNC$,POS$/
C***      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
C***      FORMAT('TRACE ',I3,' POS$ : ',110A1)
C***      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C***      IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000
C***
C***      IF(LB.LE.(LA-LSTART+1))GOTO 10
C***      POS$=0
C***      IF(IPRT.GE.20) WRITE(6,201)
C***      FORMAT('FIRST STRING TOO SHORT TO CONTAIN SECOND STRING')
C***      RETURN
C***
C***      LEFT=LA-LB+1
C***      DO 25 I=LSTART,LEFT
C***      DO 20 J=1,LB
C***      JN=I+J-1
C***      IF(IPRT.GE.20)WRITE(6,202)JN,A$(JN),J,B$(J)
C***      FORMAT('COMPARE A$(',I3,',)=',A1,' WITH B$(',I3,',)=',A1)
C***      IF(A$(JN).NE.B$(J))GOTO 25
C***      CONTINUE
C***      POS$=I
C***      IF(IPRT.GE.20)WRITE(6,203) I
C***      FORMAT('AT POSITION',I3,' SECOND STRING FOUND IN FIRST')
C***      RETURN
C***
C***      CONTINUE
C***
C***      ROUTINE WILL FALL THROUGH TO FOLLOWING IF NO MATCH FOUND.
C***
C***      PCS$=0
C***      IF(IPRT.GE.20)WRITE(6,204)

```


POS000490
 POS000500
 POS000510
 POS000520
 POS000530
 POS000540
 POS000550
 POS000560
 POS000570
 POS000580

```

204  FORMAT(' FIRST STRING SEARCHED AND SECOND STRING NOT FOUND.')
      RETURN
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      POS$=-1
      RETURN
      END
  
```



```

C**      INTEGER FUNCTION SEG$(A$,LA,B$,LB,LSTART,LOUT)
C**      *****
C**      A SUBSTRING IS EXTRACTED FROM A$ AND ASSIGNED TO B$. THE
C**      SUBSTRING IN A$ BEGINS AT POSITION LSTART AND IS LOUT CHAR-
C**      ACTERS LONG.
C**
C**      IF EITHER A$ IS THE NULL STRING
C**      OR A$ DOES NOT HAVE ENOUGH CHARACTERS
C**      THEN B$ IS RIGHT PADDED WITHBLANKS SO IT HAS LOUT CHAR
C**
C**      IF LOUT=0, THEN B$ BECOMES THE NULL STRING
C**
C**      THE ACTUAL NUMBER OF NON-BLANK CHARACTERS OBTAINED FROM A$
C**      IS RETURNED AS THE VALUE OF THE FUNCTION SEG$.
C**      *****
C**      IMPLICIT INTEGER(A-Z)
C**      COMMON/TEXT/IDIM,IPRT
C**      INTEGER*2 A$(IDIM),B$(IDIM)
C**      INTEGER*2 BLNK//
C**      INTEGER*4 FUNC$/SEG,/
C**      LOGICAL FIRST
C**      IF(IPRT,GE,10) WRITE(6,200)LA,(A$(I),I=1,LA)
C**      FORMAT(1,TRACE,13,1,SEG$,110A1)
C**      FIRST=.TRUE.
C**      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C**      IF(LSTART.GT.IDIM.OR.LSTART.LT.0) GOTO 6000
C**      IF(LOUT.GT.IDIM.OR.LOUT.LT.0) GOTO 6000
C**
C**      IF(LOUT.NE.0) GOTO 5
C**      LB=LOUT
C**      IF(IPRT,GE,20) WRITE(6,202)
C**      FORMAT(1, NULL STRING ASSIGNED BY SEG$,1)
C**      SEG$=0
C**      RETURN
C**
C**      IF(LSTART.LE.LA) GOTO 15
C**      DO 10 I=1,LOUT
C**      B$(I)=BLNK
C**      CONTINUE
C**      LB=LOUT
C**      IF(IPRT,GE,20) WRITE(6,201) LSTART,LOUT
C**      FORMAT(1,START POINT (1,I3,1) GREATER THAN LENGTH FIRST STRING,1
200
C
C
202
C
C
5
10
201

```



```

2      SEG$=0      /,'      NEW STRING HAS',I3,' BLANKS')
      RETURN
C
C
C
15    DO 35 I=1,LOUT
      IM=I
      IF(I.GT.(LA-LOUT+1))GOTO 25
      B$(I)=A$(LSTART+I-1)
      GOTO 35
25    B$(I)=BLNK
      IF(.NOT.FIRST) GOTO 35
      FIRST=.FALSE.
      SEG$=IM-1
35    CONTINUE
      IF(FIRST) SEG$=IM
      IF(IPRT.GE.20) WRITE(6,203) SEG$
      FORMAT(' SEG$ OBTAINED ',I3,' CHARACTERS FROM FIRST STRING')
      LB=LOUT
      RETURN
C
C
C
C
      ERROR HANDLING SECTION FOLLOWS
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      SEG$=-1
      RETURN
      END

```

```

SEG000490
SEG000500
SEG000510
SEG000520
SEG000530
SEG000540
SEG000550
SEG000560
SEG000570
SEG000580
SEG000590
SEG000600
SEG000610
SEG000620
SEG000630
SEG000640
SEG000650
SEG000660
SEG000670
SEG000680
SEG000690
SEG000700
SEG000710
SEG000720
SEG000730
SEG000740
SEG000750
SEG000760
SEG000770
SEG000780
SEG000790

```



```

C** INTEGER FUNCTION TRIM$(A$,LA)
C**
C** THIS FUNCTION STRIPS A STRING OF LEADING AND TRAILING BLANKS.
C**
C** THE RETURN VALUE OF THE FUNCTION IS SET TO THE LENGTH OF THE
C** STRING REMAINING AFTER THE BLANKS ARE REMOVED.
C**
C**
C** IMPLICIT INTEGER(A-Z)
C** COMMON/TEXT/IDIM,IPRT
C** INTEGER*2 A$(IDIM)
C** INTEGER*2 BLNK//
C** INTEGER*4 FUNC$/TRIM//
C** IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
C** FORMAT(1,TRACE,13,1,TRIM$,
C** IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C**
C** CHECK A$ FOR TRAILING BLANKS
C**
C** IM=0
C** DO 60 I=1,LA
C** INDEX=LA-I+1
C** IF(A$(INDEX).NE.BLNK) GOTO 65
C** IM=IM+1
C** CONTINUE
C** IF(IM.EQ.0) GOTO 70
C** IF(IPRT.GE.20) WRITE(6,207) IM
C** FORMAT(1,FOUND,13,1,TRAILING BLANKS IN STRING')
C** IF(IM.NE.LA) GOTO 70
C** LA=0
C** TRIM$=0
C** IF(IPRT.GE.20) WRITE(6,208)
C** FORMAT(1,FOUND STRING IS ALL BLANKS')
C** RETURN
C** IEND=LA-IM
C**
C** CHECK A$ FOR LEADING BLANKS
C**
C** IM=0
C** DO 10 I=1,IEND
C** IF(A$(I).NE.BLNK) GOTO 15

```


10	IM=IM+1	TR100490
15	CONTINUE	TR100500
	IF(IM.EQ.0) GOTO 25	TR100510
211	IF(IPRT.GE.20) WRITE(6,211) IM	TR100520
25	FORMAT(1, FOUND, I3, ' LEADING BLANKS IN STRING')	TR100530
	IBEG=1+IM	TR100540
		TR100550
		TR100560
		TR100570
		TR100580
		TR100590
		TR100600
		TR100610
		TR100620
		TR100630
216	IF(IPRT.GE.10) WRITE(6,216) LOSS	TR100640
	FORMAT(1, STRING TOO LONG FOR MAX STRING LENGTH. LOST, I3)	TR100650
	LA=IDIM	TR100660
	IEND=IEND-LOSS	TR100670
		TR100680
		TR100690
		TR100700
		TR100710
30	INDEX=1	TR100720
	DO 85 I=IBEG, IEND	TR100730
	A\$(INDEX)=A\$(I)	TR100740
	IF(IPRT.GE.30) WRITE(6,209) I, A\$(I), INDEX, A\$(INDEX)	TR100750
209	FORMAT(1, MOVE A\$(, I3, ')=, A1, ' IS NOW A\$(, I3, ')=, A1)	TR100760
	INDEX=INDEX+1	TR100770
85	CONTINUE	TR100780
		TR100790
		TR100800
		TR100810
		TR100820
		TR100830
		TR100840
222	IF(IPRT.GE.10) WRITE(6,222) LA, (A\$(I), I=1, LA)	TR100850
	FORMAT(1, TRIM\$, I3, ' AFTER : ', 11) A1)	TR100860
	IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000	TR100870
		TR100880
		TR100890
	TRIM\$=LA	TR100900
	RETURN	TR100910
		TR100920
		TR100930
		TR100940
6000	WRITE(6,6001) FUNC\$	TR100950
6001	FORMAT(1, *** STRING LENGTH ERROR *** , A4)	TR100960
	WRITE(6,6010) LA, LB, IDIM	

TRI00970
TRI00980
TRI00990
TRI01000

LDIM=,I10)

LB=,I10,

LA=,I10,

FCRMAT(, LA=,I10,
TRIM\$=-1
RETURN
END

6010

[illegible]

C 35	INDEX=INDEX+101	XEQ00970
	GOTO 75	XEQ00980
C		XEQ00990
C		XEQ01000
C		XEQ01010
C		XEQ01020
C		XEQ01030
C		XEQ01040
C		XEQ01050
C 50	OPERAND MUST BE A NUMERIC LOCAL LABEL	XEQ01060
C	IF(IVAL\$(A\$,LA,INDEX))6015,75,75	XEQ01070
C		XEQ01080
C		XEQ01090
C		XEQ01100
C		XEQ01110
C		XEQ01120
C 75	PROCESS THE GOTO INSTRUCTION (OPERAND>14)	XEQ01130
	M(M1)=IBYTE	XEQ01140
	IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE	XEQ01150
C 210	FORMAT(, XEQ\$,15,, LENGTH OF NEXT INSTR IS',I3)	XEQ01160
	M1=M1+1	XEQ01170
C		XEQ01180
C		XEQ01190
C		XEQ01200
C		XEQ01210
	LOAD THE XEQ INSTR PREFIX	XEQ01220
	M(M1)=PREFIX	XEQ01230
C 211	IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)	XEQ01240
	FORMAT(, XEQ\$,15,, XEQ PREFIX	XEQ01250
	M1=M1+1	XEQ01260
C		XEQ01270
C		XEQ01280
C	LOAD THREE BYTE GOTO INSTR NULL INSTR (POSITION HOLDER FOR POINTER	XEQ01290
		XEQ01300
	IF(INDIR) GOTO 95	XEQ01310
	M(M1)=0	XEQ01320
	IF(IPRT.GE.10)WRITE(6,221)M1,M(M1)	XEQ01330
C 221	FORMAT(, XEQ\$,15,, NULL FOR XEQ INSTR',T75,I3)	XEQ01340
	M1=M1+1	XEQ01350
C		XEQ01360
C		XEQ01370
C	LOAD THE 2D OPERAND OF THE XEQ INSTR	XEQ01380
C		XEQ01390
C		XEQ01400
C 95	IF(INDIR) INDEX=INDEX+128	XEQ01410
	NOTE THAT FOR XEQ IND THE HIGH ORDER BIT IS SET	XEQ01420
	M(M1)=INDEX	XEQ01430
C	IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)	XEQ01440

[illegible]

XEQ001930
XEQ001940
XEQ001950
XEQ001960
XEQ001970
XEQ001980
XEQ001990
XEQ002000
XEQ002010
XEQ002020
XEQ002030
XEQ002040
XEQ002050
XEQ002060
XEQ002070
XEQ002080
XEQ002090
XEQ002100
XEQ002110
XEQ002120
XEQ002130
XEQ002140
XEQ002150
XEQ002160
XEQ002170
XEQ002180
XEQ002190
XEQ002200
XEQ002210
XEQ002220
XEQ002230
XEQ002240
XEQ002250
XEQ002260
XEQ002270
XEQ002280
XEQ002290
XEQ002300
XEQ002310
XEQ002320
XEQ002330
XEQ002340
XEQ002350
XEQ002360
XEQ002370

```

214 IF(IPRT,GE,10)WRITE(6,214)M1,M(M1)
    FORMAT(, XEQ$,15, ALPHA XEQ INSTR',T75,I3)
    M1=M1+1

    SET INDICATOR FOR NUMBER OF ALPHA CHARS IN LABEL

    U=240
    FOR GTO U=240      FOR LBL U=241      FOR XEQ U=240
    M(M1)=U+LENGTH
    IF(IPRT,GE,10)WRITE(6,216)M1,M(M1)
    FORMAT(, XEQ$,15, LENGTH CODE ALPHA XEQ',T75,I3)
    M1=M1+1

    ADD ALPHABETIC CHARACTERS AND RETURN

    LA=LENGTH+1
    XEQ$=ALPH$(A$,LA,M,M1)
    RETURN

    ERROR HANDLING SECTION FOLLOWS

    WRITE(6,6001) FUNC$
    FORMAT(, ***, STRING LENGTH ERROR ***, A4)
    WRITE(6,6010) LA, LB, IDIM      LB=', I10, '      IDIM=', I10)
    FORMAT(, LA=', I10, '
    XEQ$=-1
    RETURN
    WRITE(6,6016)
    FORMAT(, ***) INVALID SECOND OPERAND IN XEQ INSTR *****)
    XEQ$=-1
    RETURN
    WRITE(6,6021)
    FORMAT(, ***) FOUND THREE OPERANDS, EXPECTING IND *****)
    XEQ$=-1
    RETURN
    END

```



```

C** INTEGER FUNCTION XRO$(A$,LA,M,M1)
C**
C** STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN XROM INSTRUCTION.
C**
C** THE XROM INSTRUCTIONS ARE SUBROUTINE CALLS TO HP SUPPLIED
C** ROM ENCODED SUBROUTINES. THE SECOND AND THIRD OPERANDS MUST
C** BE NUMERIC.
C**
C**
C** THE RETURN VALUE OF THE FUNCTION XRO$ IS SET AS FOLLOWS:
C**      ) = CONTINUE TO COMPILE
C**    -1 = AN ERROR IN COMPILING THE INSTRUCTION.
C**
C** IMPLICIT INTEGER(A-Z)
C** COMMON/TEXT/IOIM,IPRT
C** INTEGER*2 A$(IDIM),
C** INTEGER*2 COMMA/,/,SS1$(40)
C** INTEGER*4 FUNC$/XRO:/
C** INTEGER*4 M(1)
C** REAL*4 RFRAC,RDIF
C** IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
C** FORMAT('TRACE ',I3,' XRO$ :',110A1)
C** IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C**
C** 200 C
C** C
C** C
C** IF(LCUT$(A$,LA,4)) 6015,6020,7
C** IF(TRIM$(A$,LA)) 6015,6020,10
C** P1=POS$(A$,LA,COMMA,1,1)
C** P2=P1-1
C** IF(SEG$(A$,LA,SS1$,LSS1,1,P2))6015,6020,12
C** IF(LCUT$(A$,LA,PI)) 6015,6020,15
C** IF(IIVAL$(SS1$,LSS1,ROM)) 6030,20,20
C** IF(IIVAL$(A$,LA,PGM)) 6030,25,25
C** IFRAC=ROM/4
C** IFRST=160+IFRAC
C** RFRAC=FLOAT(ROM)/4.0
C** RDIF=RFRAC-FLOAT(IFRAC)
C** RDIF=256.0*RDIF
C** IDIF=INT(RDIF)
C** ISECOND=PGM+IDIF

```

THE RETURN VALUE OF THE FUNCTION XRD\$ IS SET AS FOLLOWS:

0	=	CONTINUE TO COMPILE
-1	=	AN ERROR IN COMPILING THE INSTRUCTION.

[illegible]


```

6021      FORMAT(' ***** INVALID PROGRAM NUMBER IN XRC INSTR *****')
        XRC$=-1
        RETURN
6030      WRITE(6,6031)
6031      FORMAT(' ***** NUMERIC CONVERSION ERROR IN XRC INSTR *****')
        XRC$=-1
        RETURN
        END

```

```

XR0000970
XR0000980
XR0000990
XR0001000
XR0001010
XR0001020
XR0001030
XR0001040

```



```

C      CSPACE=0.08*HEIGHT
      IS THE HEIGHT OF THE TITLE CHARACTERS.
C      TSIZE=0.22*HEIGHT
      IS THE HEIGHT OF THE SPACE BETWEEN THE TITLE AND THE
C      TSPACE
      FIRST BARCODE ROW NUMBER.
C      TSPACE=CSIZE+(1.5*CSPACE)
      IS THE RELATIVE HEIGHT OF THE SUM OF THE BAR HEIGHT
      PLUS THE SPACE BETWEEN THE BARS, INCLUDING LABELS
C      HFACTR=1.7*HEIGHT
      IS THE HEIGHT OF EACH PAGE OF BARCODE
C      PLONG=11.0
      IS THE WIDTH OF EACH PAGE CF BARCODE
C      PWIDE=8.5
      IS THE HEIGHT OF THE TOP MARGIN OF THE PAGE
C      TMAR=1.0
      IS THE HEIGHT OF THE MARGINS REPRESENTS HOW
C      ESPACE
      FAR THE PAPER IS ADVANCED BETWEEN PAGES
C      ESPACE=TMAR+1.5
      IS THE HEIGHT OF THE LEFT SIDE MARGIN ON THE PAGE
C      SMAR=1.5
      IS A LEFT MARGIN OFFSET USED TO MOVE THE PHYSICAL
C      ZERO
      PLOT AWAY FROM THE LEFT MARGIN OF THE PLOTTER
C      ZERO=3.0
      IS THE NUMBER OF BARCODE ROWS THAT WILL BE DRAWN PER PAGE
C      PERPGE
      PERPGE=IFIX((PLONG-ESPACE)/HFACTR)
      CALL PLOTS(,0,0)
      CALL PLOT(0.0,0.0,3)
      IROW=0
      IPRT=0
      X=TMAR
      Y=ZERO

C      READ THE TITLE OF THE BARCODE PROGRAM FROM THE INPUT FILE
C      READ(5,103)(TITLE(I),I=1,80)
C      FORMAT(80A1)
C
C      READ THE ROW OF ZERO'S AND ONE'S FROM THE INPUT FILE
C      READ(5,101,END=3000)(IN(I),I=1,132)
C      FORMAT(66A1,/,5X,66A1)
C      IF(IPRT.GE.20) WRITE(6,201) (IN(I),I=1,132)
C      FORMAT(,132A1)
C
103
C
C
C
C
101
C
101
C
201
C

```

```

VER00490
VER00500
VER00510
VER00520
VER00530
VER00540
VER00550
VER00560
VER00570
VER00580
VER00590
VER00600
VER00610
VER00620
VER00630
VER00640
VER00650
VER00660
VER00670
VER00680
VER00690
VER00700
VER00710
VER00720
VER00730
VER00740
VER00750
VER00760
VER00770
VER00780
VER00790
VER00800
VER00810
VER00820
VER00830
VER00840
VER00850
VER00860
VER00870
VER00880
VER00890
VER00900
VER00910
VER00920
VER00930
VER00940
VER00950
VER00960

```


C
C
20

```

WRITE THE TITLE ON THE PLOT
      IF (MOD(IROW, PERPGE).NE.0) GOTO 30
      X=X+ESPACE
      CALL NEWPEN(2)
      XM1=X-TMAR
      XM2=X+PLONG-TMAR
      YM1=Y-SMAR
      YM2=Y+PWIDE-SMAR
      CALL PLOT(XM1, YM1, 3)
      CALL PLOT(XM2, YM1, 2)
      CALL PLOT(XM2, YM2, 2)
      CALL PLOT(XM1, YM2, 2)
      CALL PLOT(XM1, YM1, 2)
      CALL PLOT(X, Y, 3)
      CALL SYMBOL(X, Y, TSIZE, TITLE, 90.0, 72)
      CALL NEWPEN(NIBS)
      X=X+TSPACE
      Y=ZERO
      CALL PLOT(X, Y, 3)

```

C
C
C
C
30

```

LABEL THE BAR CODE ROW
      ROWNUM=RCWNUM+1.0
      IROW=IROW+1
      CALL NEWPEN(2)
      CALL NUMBER(X, Y, CSIZE, ROWNUM, 90.0, -1)
      CALL NEWPEN(NIBS)
      X=X+CSPACE
      CALL PLOT(X, Y, 3)

```

C
C
C

```

CCONVERT THE ZERO'S AND ONE'S INTO BARS OF CORRECT WIDTH
DC 1000 I=1, 132

```

C
C
C
C
C
50

```

CHECK FOR ZERO OR ONE BIT
      IF(IN(1).EQ.ZERO) GOTO 100
      IF(IN(1).EQ.ONE) GOTO 200
      IF(IN(1).EQ.BLNK) GOTO 2000

```

C
C

VER00970
VER00980
VER00990
VER01000
VER01010
VER01020
VER01030
VER01040
VER01050
VER01060
VER01070
VER01080
VER01090
VER01100
VER01110
VER01120
VER01130
VER01140
VER01150
VER01160
VER01170
VER01180
VER01190
VER01200
VER01210
VER01220
VER01230
VER01240
VER01250
VER01260
VER01270
VER01280
VER01290
VER01300
VER01310
VER01320
VER01330
VER01340
VER01350
VER01360
VER01370
VER01380
VER01390
VER01400
VER01410
VER01420
VER01430
VER01440

DRAW A ZERO BAR OF UNIT WIDTH

X=X+HEIGHT
CALL PLOT(X,Y,2)
Y=Y+DOUBLE
X=X-HEIGHT
CALL PLOT(X,Y,3)
GOTO 1000

DRAW A ONE BAR OF DOUBLE UNIT WIDTH

X=X+HEIGHT
CALL PLOT(X,Y,2)
Y=Y+UNIT
X=X-HEIGHT
CALL PLOT(X,Y,3)
X=X+HEIGHT
CALL PLOT(X,Y,2)
Y=Y+DOUBLE
X=X-HEIGHT
CALL PLOT(X,Y,3)
GOTO 1000

CONTINUE

GOTO NEXT ROW

X=X+HFACTR
Y=ZERO
CALL PLOT(X,Y,3)
GOTO 10

FINISH UP THE PLOT: PROGRAM IS CCMPLETE

CALL PLOT(0.,0.,+999)
STOP
END

VER01450
VER01460
VER01470
VER01480
VER01490
VER01500
VER01510
VER01520
VER01530
VER01540
VER01550
VER01560
VER01570
VER01580
VER01590
VER01600
VER01610
VER01620
VER01630
VER01640
VER01650
VER01660
VER01670
VER01680
VER01690
VER01700
VER01710
VER01720
VER01730
VER01740
VER01750
VER01760
VER01770
VER01780
VER01790
VER01800
VER01810
VER01820
VER01830
VER01840
VER01850
VER01860
VER01870
VER01880
VER01890
VER01900
VER01910
VER01920

THE FOLLOWING IS THE DATA SET READ BY THE CROSS COMPILER.

0	62				
HP41C	SOURCE	CCDE:			
+	64	7	1	1	1
-	65	8	6	1	1
*	66	6	5	1	1
/	67	9	7	1	1
X<Y?	68	97	92	4	1
X>Y?	69	101	95	4	1
X<=Y?	70	94	90	5	1
E+	71	3	2	2	1
E-	72	4	3	2	1
HMS+	73	50	48	4	1
HMS-	74	51	49	4	1
MOD	75	60	58	3	1
%	76	1	8	1	1
1/Ct	77	2	9	3	1
P-R	78	63	61	3	1
R-P	79	68	67	3	1
LN	80	56	54	2	1
X1/2	81	102	94	3	1
SQRT	82	30	78	4	1
Y1/X	83	103	101	3	1
CHS	84	25	23	3	1
E1/X	85	39	35	3	1
LOG	86	58	56	3	1
101/X	87	11	103	4	1
E1/X-1	88	40	38	5	1
SIN	89	79	77	3	1
CCS	90	32	30	3	1
TAN	91	87	85	3	1
ASIN	92	19	17	4	1
ACCS	93	13	11	4	1
ATAN	94	21	19	4	1
DEC	95	34	32	3	1
1/X	96	10	102	3	1
ABS	97	12	10	3	1
FACT	98	41	39	4	1
X#C?	99	90	98	4	1
X>C?	100	100	96	4	1
LN1+X	101	57	55	5	1
X<C?	102	92	93	4	1
X=C?	103	98	100	4	1
INT	104	53	51	3	1
FRC	105	45	43	3	1
D-R	106	32	31	3	1
R-C	107	67	66	3	1
HMS	108	49	47	3	1
HR	109	52	50	2	1
RNC	110	72	71	3	1
QCT	111	61	59	3	1
CL&	112	26	24	3	1
X<>Y	113	96	39	4	1
PI	114	64	62	2	1
CLST	115	30	28	4	1
RI	116	74	65	2	1
RDN	117	71	70	3	1
LASTX	118	55	53	5	1

CLX	119	31	29	3	1
X=Y?	120	99	99	4	1
X#Y?	121	91	97	4	1
SIGN	122	78	75	4	1
X<=C?	123	93	91	5	1
MEAN	124	55	57	4	1
SDEV	125	76	74	4	1
AVIEW	126	22	20	5	1
CLC	127	28	26	3	1
DEG	128	35	33	3	1
RAD	129	69	68	3	1
GRAD	130	48	46	4	1
ENTER I	131	38	37	6	1
STC P	132	86	84	4	1
RTN	133	73	72	3	1
BEEP	134	23	21	4	1
CLA	135	27	25	3	1
AST F	136	18	16	4	1
PSE	137	66	64	3	1
CLRG	138	25	27	4	1
ACFF	139	15	13	4	1
ACN	140	16	14	3	1
OFF	141	62	60	3	1
FRMPT	142	65	63	6	1
ADV	143	14	12	3	1
RCL	144	70	69	3	2
STC	145	85	83	3	2
ST+	146	82	79	3	2
ST-	147	83	81	3	2
ST*	148	81	80	3	2
ST/	149	84	82	3	2
ISG	150	54	52	3	2
DSE	151	36	34	3	2
VIEW	152	89	87	4	2
&REG	153	5	4	4	2
ASTC	154	20	18	4	2
ARCL	155	17	15	4	2
FIX	156	44	42	3	2
SCI	157	75	73	3	2
ENG	158	37	36	3	2
TONE	159	88	86	4	2
XRCM	160			4	2
XRCM	161			4	2
XRCM	162			4	2
XRCM	163			4	2
XRCM	164			4	2
XRCM	165			4	2
XRCM	166			4	2
XRCM	167			4	2
SF	168	77	75	2	2
CF	169	24	22	2	2
FS?C	170	47	45	4	2
FC?C	171	43	41	4	2
FS?	172	46	44	3	2
FC?	173	42	40	3	2
X<>	206	55	88	3	2

LIST OF REFERENCES

1. Wickes, W. C., Synthetic Programming on the HP41C, Larkin Publications, P.O. Box 987, College Park, Maryland 20740, 1980.
2. Hewlett-Packard Corporation, The HP41C/41CV Alphanumeric Full Performance Programmable Calculator Owner's Handbook and Programming Guide, 1980.
3. Dahl, O. J., Dijkstra, E. W., and Hoare, C. A. R., Structured Programming, Academic Press, 1972.
4. Hamming, R. W., The Art of Programming the TI-59, instructional monograph used at the Naval Postgraduate School, April 1980.
5. Weir, M. D., Calculator Clout, Prentice Hall, 1981.
6. Knuth, D.E., The Art of Computer Programming, Volume 3, Addison-Wesley, 1973.
7. Mendenhall, W., Scheaffer, R. L., and Wackerly, D. D., Mathematical Statistics with Applications, Duxbury Press, 1981.
8. Hillier, F. S. and Lieberman, G. J., Introduction to Operations Research, Holden-Day, 1980.
9. U. S. Army Human Engineering Laboratory, Technical Memorandum 1-76, A Compiler for Pocket Calculators, by D. B. Blazie, January 1976.
10. Carvalho, C. J., "BASIC Barcode Program," PPC Calculator Journal, v. 8, n. 2, March-April 1981.
11. McNeal, T., "Generating Bar Code in the Hewlett-Packard Format," BYTE, v. 6, n. 1, January 1981.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
4. Associate Professor S. H. Parry, Code 55Py Department of Operations Research Naval Postgraduate School Monterey, California 93940	5
5. Associate Professor R. H. Shudde, Code 55Su Department of Operations Research Naval Postgraduate School Monterey, California 93940	5
6. Office of the Commanding General U. S. Readiness Command ATTN: General Donn A. Starry MacDill AFB, Florida 33621	1
7. Office of the Commanding General U. S. Army Training and Doctrine Command ATTN: General Glenn Otis Fort Monroe, Virginia 23651	1
8. Chief TRADOC Research Element Monterey Naval Postgraduate School Monterey, California 93940	1
9. Headquarters U. S. Army Training and Doctrine Command ATTN: Director, Studies and Analysis Directorate Mr. S. Goldberg Fort Monroe, Virginia 23651	1
10. Headquarters U. S. Army Training and Doctrine Command ATTN: ATCG-T (BG Morelli) Fort Monroe, Virginia 23651	1

11. Headquarters 1
U. S. Army Training and Doctrine Command
ATTN: Director, Maneuver Directorate Combat
Developments (LTC John VanZant)
Fort Monroe, Virginia 23651
12. Headquarters 1
U. S. Army Training and Doctrine Command
ATTN: ATTG-AE (LTC Smith)
Fort Monroe, Virginia 23651
13. Mr. Walter Hollis 1
Deputy Under Secretary of the Army
(Operations Research)
Department of the Army
Washington, D. C. 20310
14. LTG Howard Stone 1
Commanding General
U.S. Army Combined Arms Center
Fort Leavenworth, Kansas 66027
15. Director 1
Combined Arms Combat Development Activity
ATTN: ATZL-CAC-A (Mr. Lee Pleger)
Fort Leavenworth, Kansas 66027
16. Director, BSSD 1
Combined Arms Combat Development Activity
ATTN: ATZLCA-DS
Fort Leavenworth, Kansas 66027
17. Director 1
Combat Analysis Office
ATTN: Mr. Kent Pickett
Fort Leavenworth, Kansas 66027
18. Commandant 1
U. S. Army Command and General Staff College
ATTN: Education Advisor
Room 123, Bell Hall
Fort Leavenworth, Kansas 66027
18. Commandant 2
U. S. Army Command and General Staff College
ATTN: ATZL-SWB (MAJ Witschonke)
Fort Leavenworth, Kansas 66027
19. Dr. Wilbur Payne, Director 1
U. S. Army TRADOC Systems Analysis Activity
White Sands Missile Range, New Mexico 88002
20. Headquarters, Department of the Army 1
Office of the Deputy Chief of Staff
for Operations and Plans
ATTN: DAMO-2D
Washington, D.C. 20310

21. Commander 1
U. S. Army Concepts Analysis Agency
ATTN: MOCA-WG (LTC Earl Darden)
8120 Woodmont Avenue
Bethesda, Maryland 20014
22. Director 1
U. S. Army Material Systems Analysis Activity
ATTN: DRXSY-CM (Mr. Bill Niemeyer)
Aberdeen Proving Grounds, Maryland 21005
23. Director 1
U. S. Army Night Vision and Electro-Optical Lab
ATTN: DEL-NV-VI (Mr. Frank Shields)
Fort Belvoir, Virginia 22060
24. Director 1
U. S. Army TRADOC Systems Analysis Activity
ATTN: Mr Ray Heath
White Sands Missile Range, New Mexico 88002
25. Director 1
Combat Developments Studies Division
ATTN: MAJ W. Scott Wallace
U. S. Army Armor Agency
Fort Knox, Kentucky 40121
26. Commandant 1
U. S. Army Field Artillery School
ATTN: ATSF-MBT (CPT Steve Starner)
Fort Sill, Oklahoma 73503
27. Director 1
Combat Developments
U. S. Army Aviation Agency
Fort Rucker, Alabama 36362
28. Director 1
Combat Developments
U. S. Army Infantry School
Fort Benning, Georgia 31905
29. Director 1
Armored Combat Vehicle Technology Program
ATTN: COL Fleming
U. S. Army Armor Center
Fort Knox, Kentucky 40121
30. Director 1
Combat Developments
ATTN: MAJ William D. Meiers
U. S. Army Air Defense Agency
Fort Bliss, Texas 79905
31. Commander 1
U. S. Army Logistics Center
ATTN: ATCL-OS (Mr. Cammeron/CPT Schuessler)
Fort Lee, Virginia 23801

32. Director 1
U. S. Army Material Systems Analysis Agency
ATTN: DRXSY-AA (Mr. Tom Coyle)
Aberdeen Proving Grounds, Maryland 21005
33. Deputy Chief of Staff for Combat Developments 1
U. S. Army Training and Doctrine Command
ATTN: (MG Dick Boyle)
Fort Monroe, Virginia 23651
34. Deputy Commanding General 1
Combined Arms Combat Development Activity
ATTN: ATZL-CA-DC
Fort Leavenworth, Kansas 66027
35. Commandant 1
U. S. Army Signal School
Fort Gordon, Georgia 30905
36. Associate Professor Arthur L. Schoenstadt, Code 53Zh 1
Department of Mathematics
Naval Postgraduate School
Monterey, California 93940
37. Associate Professor James K. Hartman, Code 55Hh 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940
38. Professor James G. Taylor, Code 55Tw 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940
39. Professor Gary K. Poock, Code 55Pk 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940
40. Professor Donald R. Barr, Code 55Bn 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940
41. Mr. Henry Horn, Editor 1
Hewlett-Packard Keynotes
Hewlett-Packard Corporation
1000 N. E. Circle Blvd.
Corvallis, Oregon 97330
42. Mr. Richard Nelson, Editor 1
PPC Journal
2541 W. Camden Place
Santa Ana, California 92704

43. Headquarters, Department of the Army
Office of the Deputy Chief of Staff for Personnel
ATTN: DAPE-MPE-SS (Captain James Richmann)
Washington, D. C. 20310

2

Thesis
R3945
c.1

Richmann

195511

A cross compiler and
programming support
system for the HP41CV.

Thesis
R3945
c.1

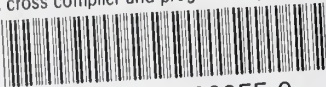
Richmann

195511

A cross compiler and
programming support
system for the HP41CV.

thesR3945

A cross compiler and programming support



3 2768 000 99055 0

DUDLEY KNOX LIBRARY